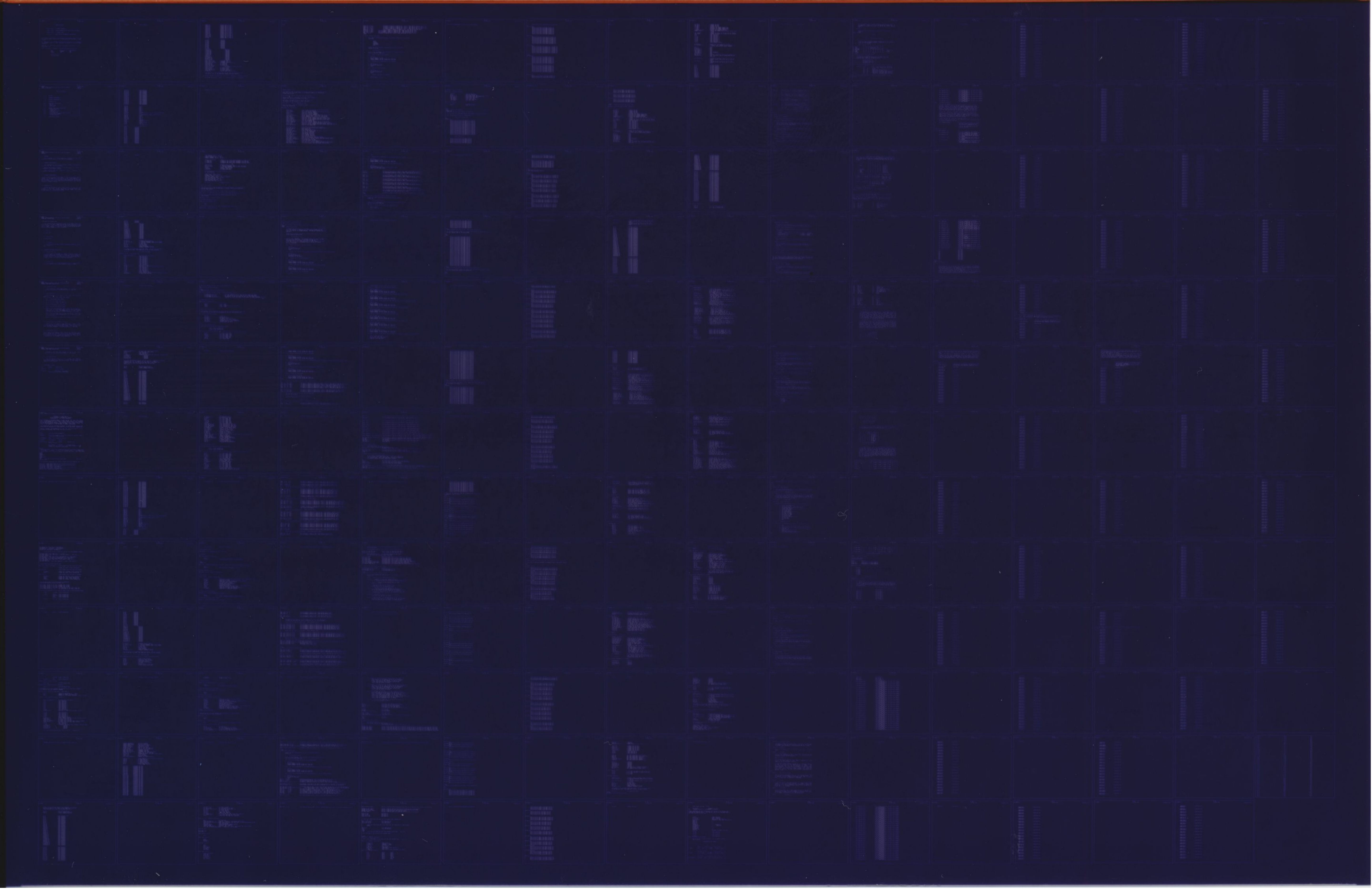
ENKCD-1.1

VAX 11/730
MICRO PART 3

EP-ENKCD-DL-1.1
1 OF 3 JAN 1986
COPYRIGHT© 1982-86
MADE IN USA



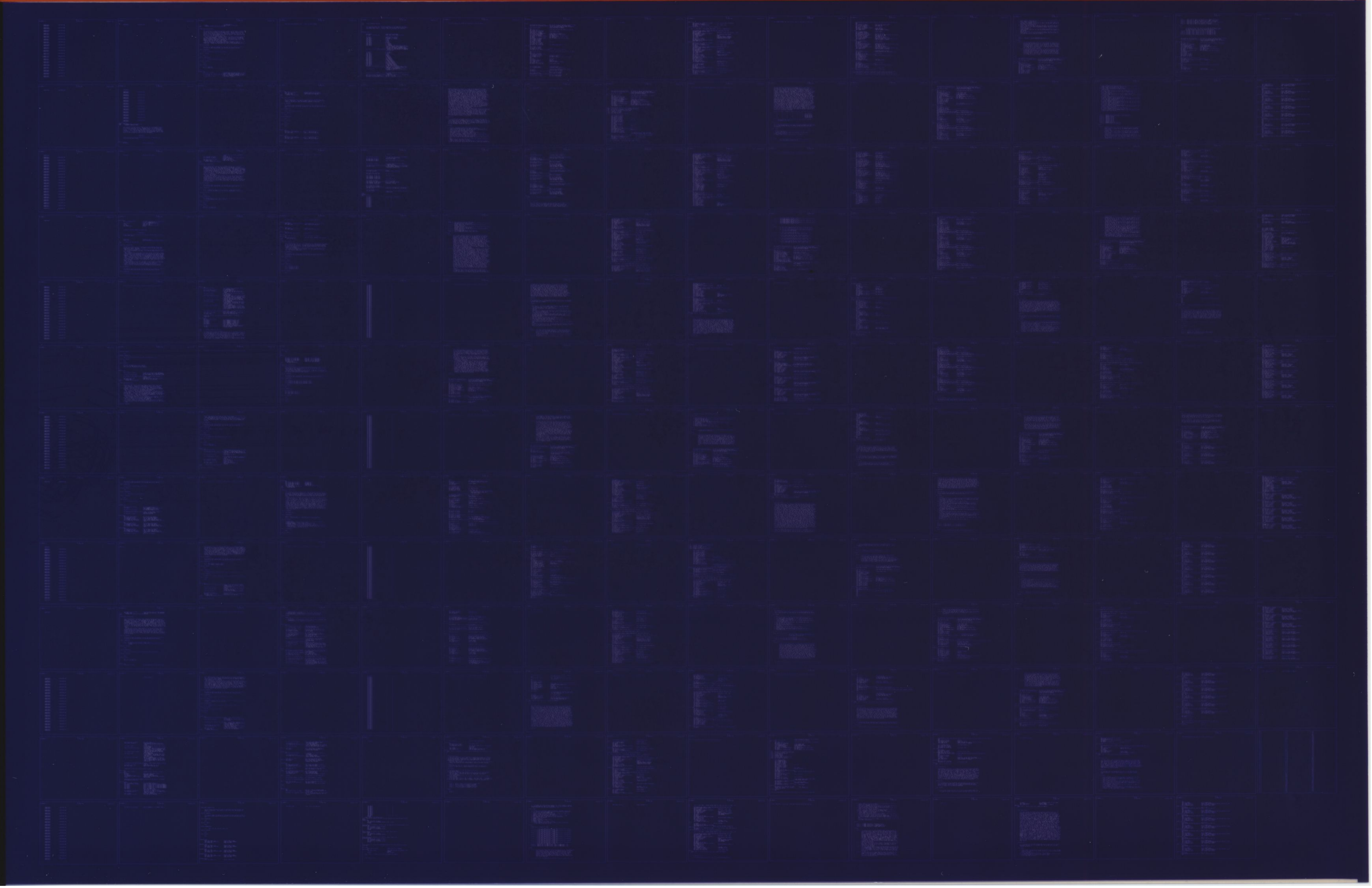
ENKCD-1.1

VAX 11/730
MICRO PART 3

EP-ENKCD-DL-1.1
2 OF 3 JAN 1986

COPYRIGHT© 1982-86

MADE IN USA



ENKCD-1.1

VAX 11/730
MICRO PART 3

EP-ENKCD-DL-1.1
3 OF 3 JAN 1986
COPYRIGHT© 1982-86

MADE IN USA

IDENTIFICATION

Product code: ZZ-ENKCD VERSION 1.1

Product name: MICRO-DIAGNOSTIC FOR VAX-11/730 CPU MODULE

Product date: 15-AUGUST-1985

Maintainer: BASE SYSTEMS DIAGNOSTIC ENGINEERING

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

Copyright (c) 1981,1985 by Digital Equipment Corporation. All Rights Reserved.

The following are trademarks of Digital Equipment Corporation.

DEC DECsystem-10 DECSYSTEM-20
DECUS MASSBUS PDP
UNIBUS VAX VMS

```
|-+-+-+-+-+-+-|
|d|i|g|i|t|a|l|
|-|-|-|-|-|-|-|
```

Table of Contents

1.0 ABSTRACT 3

2.0 HARDWARE REQUIREMENTS 3

3.0 SOFTWARE REQUIREMENTS 3

4.0 PREREQUISITES 3

5.0 OPERATING INSTRUCTIONS 4
5.1 Options 4
5.2 Event Flags 4
5.3 APT Setup 4

6.0 PROGRAM FUNCTIONAL DESCRIPTION 4
6.1 Program Overview 4
6.2 Program Size 4
6.3 Program Run Times 5
6.4 Fault Detection 5
6.5 Performance During Hardware Failures 5
6.6 Program Applications 5
6.7 Test Descriptions 6

7.0 MAINTENANCE HISTORY 6

1.0 ABSTRACT

This program is a micro-diagnostic designed to test the hardware on the CPU module of the VAX-11/730 processor.

2.0 HARDWARE REQUIREMENTS

This program will run with the minimum hardware configuration of a WCS, CPU, MCT, and array module.

Maximum main memory allowed is 5 one meg boards, for a total of 5 megabytes of main memory storage.

Other options may be installed without affecting the diagnostic's performance.

3.0 SOFTWARE REQUIREMENTS

This diagnostic must be run under the VAX-11/730 micro monitor (ENKAA) in a standalone environment. The micro-diagnostic is written into WCS RAM, wiping out any system micro-code that may have been resident. It will also use some of main memory during testing. The micro-diagnostic monitor takes up the area where the console software is normally present.

4.0 PREREQUISITES

The WCS and MCT module, and a good portion of the CPU are assumed to have been tested and known to be good before this diagnostic is run. The programs ENKBA through ENKBE, and ENKCA through ENKCC will perform this testing.

5.0 OPERATING INSTRUCTIONS

This program may be executed by typing DI SE ENKCD after the micro-monitor has been loaded and the MIC> prompt has been printed at the terminal. Both the micro-monitor and this program are loaded from a TU58 cassette or downline loaded through the APT-RD port. See the OVERVIEW MANUAL for more information.

5.1 Options

Not applicable

5.2 Event Flags

Not applicable

5.3 APT Setup

The following descibes the APT parameters needed to execute this diagnostic:
TBS

6.0 PROGRAM FUNCTIONAL DESCRIPTION

6.1 Program Overview

This program is designed to detect stuck-at faults and provide a repair tool which helps isolate the failing component. A bottom up diagnostic strategy is utilized which builds on previous tests. The tests should be run in consecutive order, to gain the best isolation of a failing component.

6.2 Program Size

This program runs in WCS RAM memory and is approximately 14K bytes long (utilizing about 4500 micro-words in WCS).

6.3 Program Run Times

This program will take approximately 21 seconds to run including initialization time and exclusive of load time.

6.4 Fault Detection

The diagnostic will report errors with the following header:

```
SECT  TST ERR   EXP   REC  OTHER  MSK  MODULE
```

1. SECT is the program name (ENKCD)
2. TST is the test number that failed.
3. ERR is the error number that failed.
4. EXP is the expected (correct) data.
5. REC is the received (actual) data.
6. OTHER is any other pertinent data (such as an address). This field is not always used. N/A will be printed under OTHER when not in use. The ERRORS section in the listing will describe the contents of this field when it is used.
7. MASK is the error mask used to check the result. Bits set to a 1 in this mask correspond to bits in the result that are not checked.
8. MODULE is the suspected module that caused the failure.

6.5 Performance During Hardware Failures

A power fail will cause a rebooting of the system. Other failures, such as a timeout or a WCS parity error, will print an error message and return to the MIC> prompt. The operator can then issue other commands, including a continue if desired.

6.6 Program Applications

This program can be used by field service to determine which module should be replaced and to verify that the replacement eliminated the failure. It can also be used as a repair tool to help isolate a failing component by manufacturing, field service or engineering. The program is APT compatible.

Customers may used the program to verify the basic integrity of the central processor module in the system.

6.7 Test Descriptions

See the actual test in the listing for detailed descriptions and error analyses. A brief description of the logic being tested by each test can be found in the table of contents of the listing.

7.0 MAINTENANCE HISTORY

DATE	VERSION	DESCRIPTION
8-MAR-82	1.0	Initial release.
22-AUG-85	1.1	Modified by Molly Bacon. Fixed problem reported in QAR 00105.

```
.TITLE  "ENKCD Micro-diagnostic for DAP module  Rev 01.10"
;
;
;                       COPYRIGHT (c) 1982,1985 BY
;                 DIGITAL EQUIPMENT CORPORATION, MAYNARD,
;                  MASSACHUSETTS.  ALL RIGHTS RESERVED.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY  BE  USED  AND  COPIED
; ONLY  IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE INCLUSION
; OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR ANY OTHER COPIES THEREOF
; MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER  PERSON.  NO
; TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
;
; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE, AND
; SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
;
; DIGITAL  ASSUMES  NO  RESPONSIBILITY  FOR  THE  USE OR RELIABILITY OF ITS
; SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
;

;++
; FACILITY:     VAX-11/730 MICRO-DIAGNOSTIC.
;
; ABSTRACT:     This micro-diagnostic tests the hardware of the DAP  module
;               in the VAX-11/730 processor.
;
; ENVIRONMENT:  ENKAA  Micro-diagnostic Monitor
;
; AUTHOR:       Mike Bibeault   7-JUL-81
;
; MODIFIED BY:  David Mayo      8-MAR-82        VERSION 01.00
;               Initial release.
;
;               Molly Bacon     8-AUG-85        Version 01.10
;               Changed Test #6, error 2, to check MASK INTERRUPT twice.
;               Problem was reported in QAR 00105.
;--
.nolist
;  MAINTAINER NOTE:  The version number must be modified before resubmitting
;  for release in 3 places.  In the .TITLE statement above, in the MODIFIED BY
;  column above, and in the micro-code at location 606 (after label LOOP.XFER).
.list
.NOCREF
.NOBIN
.SEQUENTIAL
.RTOL
.HEXADECIMAL
.PAGE
.TOC    "FIELD DEFINITIONS FOR MAIN MICRO WORD                    VER 00.08"
.SET/UPC=<000>

; THE FOLLOWING EXPRESSIONS ARE VALIDITY CHECKS FOR FIELD COMBINATIONS

.SET/A.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0]>
.SET/B.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]>
.SET/D.VAL=<.CASE[<OPC2/>]OF[0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,1]>
.SET/XD.VAL=<.EQL[<OPC1/>,<OPC1/MOVE>]>
.SET/CC.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]>
.SET/DT.VAL=<.EQL[<OPC2/>,<OPC2/MEM.REQ>]>
.SET/SCTL.VAL=<.CASE[<OPC2/>]OF[0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1]>
```

```
.SET/JCTL.VAL=<.CASE[<OPC2/>]OF[1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]>
```

```
.SET/JUMP.VAL=<.EQL[<OPC2/>,<OPC2/JUMP>]>
.SET/DECODE.VAL=<.EQL[<OPC2/>,<OPC2/DECODE>]>
.SET/MISC.VAL=<.EQL[<OPC2/>,<OPC2/MISC>]>
.SET/DP.VAL=<.EQL[<OPC/>,<OPC/BASIC>]>
;
;       THE FOLLOWING VALIDITY CHECKS ARE USE FOR THE PAGE BOUNDARY PROBLEM.
;
.SET/PAGE.PROB=<.EQL[<.AND[<7FF>,<.>]>,<7FE>]>
.SET/SKIP.LEGAL2=<.OR[<.EQL[<SCTL/>,<SCTL/RET.NOERR>]>,<.EQL[<SCTL/>,
<SCTL/POP.USTACK>]>]>
.SET/SKIP.LEGAL=<.OR[<.EQL[<SCTL/>,<SCTL/NO.SKIP>]>,<.EQL[<SCTL/>,
<SCTL/RETURN>]>,<.EQL[<SCTL/>,<SCTL/RETURN+1>]>,<SKIP.LEGAL2>]>
.SET/SKIP.PAGE.VAL=<.CASE[<PAGE.PROB>]OF[1,<SKIP.LEGAL>]>
.SET/JUMP.CHECK=<.OR[<.EQL[<JCTL/>,<JCTL/JSR>]>,<.EQL[<JCTL/>,
<JCTL/JSR.VALID>]>]>
.SET/JUMP.PAGE.VAL=<.CASE[<PAGE.PROB>]OF[1,<.XOR[1,<JUMP.CHECK>]>]>

; MICRO INSTRUCTION OPCODE DEFINITIONS
;
OPC/=<22>,.DEFAULT=<OPC/BASIC>  ; MICRO OPCODE FIELD FOR SINGLE BIT OPCODE.

        BASIC=1                 ; OPCODE FOR 'BASIC' MICRO INSTRUCTION

OPC1/=<22:20>                   ; MICRO OPCODE FIELD FOR THREE BIT OPCODES.

        EXTENDED=2              ; OPCODE FOR 'EXTENDED' MICRO INSTRUCTION
        MOVE=3                  ; OPCODE FOR 'MOVE' MICRO INSTRUCTION

OPC2/=<22:19>                   ; MICRO OPCODE FIELD FOR FOUR BIT OPCODES

        MEM.REQ=3               ; OPCODE FOR 'MEM.REQ' MICRO INSTRUCTION
        MISC=2                  ; OPCODE FOR 'MISC' MICRO INSTRUCTION
        JUMP=1                  ; OPCODE FOR 'JUMP' MICRO INSTRUCTION
        DECODE=0                ; OPCODE FOR 'DECODE' MICRO INSTRUCTION

; THE FOLLOWING FOUR FIELDS ARE RELATED TO ADDRESSING
; VARIOUS RAM MEMORIES WITHIN THE DATA PATH.
;
; THE A.ADRS  REFERS TO THE 2901 INTERNAL RAM 'A-PORT'
; THE B.ADRS  REFERS TO THE 2901 INTERNAL RAM 'B-PORT'
; THE XB.ADRS REFERS TO THE 2901 INTERNAL RAM 'B-PORT'
; THE D.ADRS  REFERS TO THE LOWER 128 LOCATIONS OF THE LOCAL STORE RAM
; THE XD.ADRS REFERS TO   ALL     256 LOCATIONS OF THE LOCAL STORE RAM
;
; THE VALIDITY STATEMENTS ALLOW USAGE OF THESE FIELDS IN THE FOLLOWING MANNER
;
;       A.ADRS          EXTENDED  MICRO INSTRUCTION
;
;       B.ADRS          BASIC     MICRO INSTRUCTION
;                       EXTENDED  MICRO INSTRUCTION
;                       MOVE      MICRO INSTRUCTION
;                       DECODE.IS MICRO INSTRUCTION
;
;       XB.ADRS         MOVE XWR  MICRO INSTRUCTION
;
```

```
;       D.ADRS          BASIC     MICRO INSTRUCTION
;                       MEM.REQ   MICRO INSTRUCTION
;
;       XD.ADRS         MOVE      MICRO INSTRUCTION
;
A.ADRS/=<10:9>,.VALIDITY=<A.VAL>

        MASK=3                  ; REGISTER BACKUP MASK

B.ADRS/=<8:7>,.VALIDITY=<B.VAL>,.DEFAULT=0

        MASK=3                  ; REGISTER BACKUP MASK

; THIS ADDRESS FIELD WILL CONTAIN THE LOW TWO BITS OF THE 2901 B ADDRESS
; THE THIRD BIT WILL BE FORCED BY HARDWARE.
;
XB.ADRS/=<8:7>,.VALIDITY=<.EQL[<OPC1/>,<OPC1/MOVE>]>

        BPC=0                   ; ADDRESS OF BEGINNING PC      WR[4]
        VA=1                    ; ADDRESS OF MEMORY REFERENCE WR[5]
        VAR=1                   ; ADDRESS OF LAST MEMORY REFERENCE. (WR[5] IN 2901).

;D.ADRS/=<15:9>,.VALIDITY=<D.VAL>,.DEFAULT=0


;       SAVED.2ND.REQUEST=0     ; MM SAVED STATE FOR REQUEST.
;       T1=1                    ; TEMP LOCATION 1
;       T2=2                    ; TEMP LOCATION 2
;       T3=3                    ; TEMP LOCATION 3
;       T4=4                    ; TEMP LOCATION 4
;       T5=5                    ; TEMP LOCATION 5
;       T6=6                    ; TEMP LOCATION 6
;       T7=7                    ; TEMP LOCATION 7
;       T8=8                    ; TEMP LOCATION 8
;       T9=9                    ; TEMP LOCATION 9
;       T10=0A                  ; TEMP LOCATION 10
;       BACKUP.CM.PC=0A         ;COMPATIBILITY MODE KEEPS ORIGINAL PC HERE.
;
;       T11=0B                  ; TEMP LOCATION 11
;       T12=0C                  ; TEMP LOCATION 12
;       T13=0D                  ; TEMP LOCATION 13
;       T14=0E                  ; TEMP LOCATION 14
;       T15=0F                  ; TEMP LOCATION 15
;       PC=10                   ; MACRO PROGRAM COUNTER
;       WR.BACKUP=11            ; BACKUP WORKING REGISTER FOR MOV MEM.DATA TO WR[]
;       SAVED.VAR=12            ; MM SAVED STATE FOR VAR
;       SAVED.DATA=13           ; MM SAVED STATE FOR DATA
;       SAVED.PTE.ADDRESS=14    ; MM SAVED STATE FOR PAGE TABLE ENTRIES ADDRESS
;       SAVED.PTE=15            ; MM SAVED STATE FOR PAGE TABLE ENTRIES
;       T16=16                  ; TEMP LOCATION 16
;       T17=17                  ; TEMP LOCATION 17
;       IE.TEMP4=18             ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION FOUR.
;       #FFFFF000=19                    ; CONSTANT
;       #FFFFFF00=1A                    ; CONSTANT
;       #FFFFFFFC=1B                    ; CONSTANT
;       #FFFFFFFF=1C                    ; CONSTANT
;
;       THIS LOCATION CONTAINS THE SOFT COPY OF THE PSL.  WHENEVER THE
;       HARDWARE VERSION OF THE PSL IS CHANGED (PSL.HW), THEN THIS
;       CHANGE IS ALSO REPORTED HERE BY THE MICROCODE.  ALL BITS OF
;       THE PSL ARE REALLY LIKE THOSE IN THIS WORD EXCEPT FOR THE
;       CONDITION CODES WHICH MUST BE OBTAINED FROM PSL.CC
;
;       PSL=1D                  ; SOFT COPY OF VAX PSL
;       VAR=1E                  ; VIRTUAL ADDRESS REGISTER
;       VAR2=1F                 ; VIRTUAL ADDRESS REGISTER
;
;
;       #1=20                   ; MASK 00000001 (HEX)
;       #2=21                   ; MASK 00000002
;       #4=22                   ; MASK 00000004
;       #8=23                   ; MASK 00000008
;       #10=24                  ; MASK 00000010
;       #20=25                  ; MASK 00000020
;       #40=26                  ; MASK 00000040
;       #80=27                  ; MASK 00000080
;       #100=28                 ; MASK 00000100
;       #200=29                 ; MASK 00000200
;       #400=2A                 ; MASK 00000400
;       #800=2B                 ; MASK 00000800
;       #1000=2C                ; MASK 00001000
;       #2000=2D                ; MASK 00002000
;       #4000=2E                ; MASK 00004000
;       #8000=2F                ; MASK 00008000
;       #10000=30               ; MASK 00010000
;       #20000=31               ; MASK 00020000
;       #40000=32               ; MASK 00040000
;       #80000=33               ; MASK 00080000
;       #100000=34              ; MASK 00100000
;       #200000=35              ; MASK 00200000
;       #400000=36              ; MASK 00400000
;       #800000=37              ; MASK 00800000
;       #1000000=38             ; MASK 01000000
;       #2000000=39             ; MASK 02000000
;       #4000000=3A             ; MASK 04000000
;       #8000000=3B             ; MASK 08000000
;       #10000000=3C            ; MASK 10000000
;       #20000000=3D            ; MASK 20000000
;       #40000000=3E            ; MASK 40000000
;       #80000000=3F            ; MASK 80000000
;
;       BIT0=20                 ; MASK 00000001
;       BIT1=21                 ; MASK 00000002
;       BIT2=22                 ; MASK 00000004
;       BIT3=23                 ; MASK 00000008
;       BIT4=24                 ; MASK 00000010
;       BIT5=25                 ; MASK 00000020
;       BIT6=26                 ; MASK 00000040
;       BIT7=27                 ; MASK 00000080
;       BIT8=28                 ; MASK 00000100
;       BIT9=29                 ; MASK 00000200
;       BIT10=2A                ; MASK 00000400
;       BIT11=2B                ; MASK 00000800
;       BIT12=2C                ; MASK 00001000
;       BIT13=2D                ; MASK 00002000
;       BIT14=2E                ; MASK 00004000
;       BIT15=2F                ; MASK 00008000
;       BIT16=30                ; MASK 00010000
;       BIT17=31                ; MASK 00020000
;       BIT18=32                ; MASK 00040000
;       BIT19=33                ; MASK 00080000
;       BIT20=34                ; MASK 00100000
;       BIT21=35                ; MASK 00200000
;       BIT22=36                ; MASK 00400000
;       BIT23=37                ; MASK 00800000
;       BIT24=38                ; MASK 01000000
;       BIT25=39                ; MASK 02000000
;       BIT26=3A                ; MASK 04000000
;       BIT27=3B                ; MASK 08000000
;       BIT28=3C                ; MASK 10000000
;       BIT29=3D                ; MASK 20000000
;       BIT30=3E                ; MASK 40000000
;       BIT31=3F                ; MASK 80000000
;
;       GPR0=40                 ; GPR 0
;       GPR1=41                 ; GPR 1
;       GPR2=42                 ; GPR 2
;       GPR3=43                 ; GPR 3
;       GPR4=44                 ; GPR 4
;       GPR5=45                 ; GPR 5
;       GPR6=46                 ; GPR 6
;       CM.SP=46                ; IN COMPATIBILITY THIS IS THE STACK POINTER.
;       GPR7=47                 ; GPR 7
;       CM.PC=47                ; IN COMPATIBILITY MODE THIS IS THE PC.
;       GPR8=48                 ; GPR 8
;       GPR9=49                 ; GPR 9
;       GPR10=4A                ; GPR 10
;       GPR11=4B                ; GPR 11
;       GPR12=4C                ; GPR 12
;       AP=4C                   ; ARGUMENT POINTER
;       GPR13=4D                ; GPR 13
;       FP=4D                   ; FRAME POINTER
;       SP=4E                   ; GPR 14
;       T0=4F                   ; TEMP LOCATION 0
;
;       ZERO=50                 ; ZERO CONSTANT
;       #3=51           ; CONSTANT
;       #5=52           ; CONSTANT
;       #6=53           ; CONSTANT
;       #7=54           ; CONSTANT
;       #9=55           ; CONSTANT
;       #B=56           ; CONSTANT
;       #C=57           ; CONSTANT
;       #D=58           ; CONSTANT
;       #E=59           ; CONSTANT
;       #F=5A           ; CONSTANT
;       #13=5B          ; CONSTANT
;       #1F=5C          ; CONSTANT
;       #34=5D          ; CONSTANT
;       #3B=5E          ; CONSTANT
;       #3F=5F          ; CONSTANT
;       #4B=60          ; CONSTANT
;       #66=61          ; CONSTANT
;       #A0=62          ; CONSTANT
;       #F0=63          ; CONSTANT
;       #FF=64          ; CONSTANT
;       #1FF=65         ; CONSTANT
;       #FFF=66         ; CONSTANT
;       #2001=67                ; CONSTANT
;       #3000=68                ; CONSTANT
;       #7F80=69                ; CONSTANT
;       #C000=6A                ; CONSTANT
;       #FFE0=6B                ; CONSTANT
;       #FFFF=6C                ; CONSTANT
;       #1F0000=6D              ; CONSTANT
;       #200001=6E              ; CONSTANT
;       #F27000=6F              ; CONSTANT
;       #3000000=70             ; CONSTANT
;       #41F0000=71             ; CONSTANT
;       #7000000=72             ; CONSTANT
;       #3FFFFE00=73            ; CONSTANT
;       #7F800000=74            ; CONSTANT
;       #7FFFFE00=75            ; CONSTANT
;       #7FFFFF0E=76            ; CONSTANT
;       #BF800001=77            ; CONSTANT
;
;       GPR.OS=78               ; HARDWARE INDEX TO GPRS
;       BIT.OS(3-0)=79          ; 16 LOCATION HARDWARE INDEX TO BIT MASKS
;       BIT.OS(4-0)=7B          ; 32 LOCATION HARDWARE INDEX TO BIT MASKS
;       OS=7C                   ; OS REGISTER
;       DEC.CON=7D              ; DECIMAL CARRIES
;       SIZE=7E                 ; SIZE REGISTER
;       MDT= 7E                 ; MEMORY REFERENCE DATA SIZE
;       INTERRUPT.VEC=7E        ; HARDWARE INTERRUPT VECTOR
;
;       THIS WORD IS THE HARDWARE CONDITION CODES.  THE CC'S CAN BE READ
;       OR ARE WRITTEN HERE.  NO OTHER BITS IN THE PSL ARE EFFECTED.
;
;       PSL.CC=7F               ; PSL CONDITION CODES

;XD.ADRS/=<16:9>,.VALIDITY=<XD.VAL>
;
;       SAVED.2ND.REQUEST=0     ; MM SAVED STATE FOR REQUEST.
;       T1=1                    ; TEMP LOCATION 1
;       T2=2                    ; TEMP LOCATION 2
;       T3=3                    ; TEMP LOCATION 3
;       T4=4                    ; TEMP LOCATION 4
;       T5=5                    ; TEMP LOCATION 5
;       T6=6                    ; TEMP LOCATION 6
;       T7=7                    ; TEMP LOCATION 7
;       T8=8                    ; TEMP LOCATION 8
;       T9=9                    ; TEMP LOCATION 9
;       T10=0A                  ; TEMP LOCATION 10
;       BACKUP.CM.PC=0A         ;COMPATIBILITY MODE KEEPS ORIGINAL PC HERE.
;       T11=0B                  ; TEMP LOCATION 11
;       T12=0C                  ; TEMP LOCATION 12
;       T13=0D                  ; TEMP LOCATION 13
;       T14=0E                  ; TEMP LOCATION 14
;       T15=0F                  ; TEMP LOCATION 15
;       PC=10                   ; MACRO INSTRUCTION PC
;       WR.BACKUP=11            ; BACKUP WORKING REGISTER FOR MOV MEM.DATA TO WR[].
;       SAVED.VAR=12            ; MM SAVED VAR LOCATION
;       SAVED.DATA=13           ; MM SAVED DATA LOCATION
;       SAVED.PTE.ADDRESS=14     ; MM SAVED PAGE TABLE ENTRY ADDRESS
;       SAVED.PTE=15             ; MM SAVED PAGE TABLE ENTRY
;       T16=16                   ; TEMP LOCATION 16
;       T17=17                   ; TEMP LOCATION 17
;       IE.TEMP4=18              ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION FOUR.
;       #FFFFF000=19                     ; CONSTANT
;       #FFFFFF00=1A                     ; CONSTANT
;       #FFFFFFFC=1B                     ; CONSTANT
;       #FFFFFFFF=1C                     ; CONSTANT
;
;       THIS LOCATION CONTAINS THE SOFT COPY OF THE PSL.  WHENEVER THE HARDWARE
;       VERSION OF THE PSL IS CHANGED (PSL.HW) THEN THIS CHANGE IS
;       REPORTED HERE.  ALL BITS OF THE PSL ARE REALLY LIKE THOSE HERE
;       EXCEPT FOR THE CONDITION CODES WHICH MUST BE OBTAINED
;       FROM PSL.CC.
;
;       PSL=1D                   ; SOFT COPY OF VAX PSL
;       VAR=1E                   ; VIRTUAL ADDRESS REGISTER
;       VAR2=1F                  ; VIRTUAL ADDRESS REGISTER
;
;
;       #1=20                    ; MASK 00000001 (HEX)
;       #2=21                    ; MASK 00000002
;       #4=22                    ; MASK 00000004
;       #8=23                    ; MASK 00000008
;       #10=24                   ; MASK 00000010
;       #20=25                   ; MASK 00000020
;       #40=26                   ; MASK 00000040
;       #80=27                   ; MASK 00000080
;       #100=28                  ; MASK 00000100
;       #200=29                  ; MASK 00000200
;       #400=2A                  ; MASK 00000400
;       #800=2B                  ; MASK 00000800
;       #1000=2C                 ; MASK 00001000
;       #2000=2D                 ; MASK 00002000
;       #4000=2E                 ; MASK 00004000
;       #8000=2F                 ; MASK 00008000
;       #10000=30                ; MASK 00010000
;       #20000=31                ; MASK 00020000
;       #40000=32                ; MASK 00040000
;       #80000=33                ; MASK 00080000
;       #100000=34               ; MASK 00100000
;       #200000=35               ; MASK 00200000
;       #400000=36               ; MASK 00400000
;       #800000=37               ; MASK 00800000
;       #1000000=38              ; MASK 01000000
;       #2000000=39              ; MASK 02000000
;       #4000000=3A              ; MASK 04000000
;       #8000000=3B              ; MASK 08000000
;       #10000000=3C             ; MASK 10000000
;       #20000000=3D             ; MASK 20000000
;       #40000000=3E             ; MASK 40000000
;       #80000000=3F             ; MASK 80000000
;
;       BIT0=20                  ; MASK 00000001
;       BIT1=21                  ; MASK 00000002
;       BIT2=22                  ; MASK 00000004
;       BIT3=23                 ; MASK 00000008
;       BIT4=24                 ; MASK 00000010
;       BIT5=25                 ; MASK 00000020
;       BIT6=26                 ; MASK 00000040
;       BIT7=27                 ; MASK 00000080
;       BIT8=28                 ; MASK 00000100
;       BIT9=29                 ; MASK 00000200
;       BIT10=2A                ; MASK 00000400
;       BIT11=2B                ; MASK 00000800
;       BIT12=2C                ; MASK 00001000
;       BIT13=2D                ; MASK 00002000
;       BIT14=2E                ; MASK 00004000
;       BIT15=2F                ; MASK 00008000
;       BIT16=30                ; MASK 00010000
;       BIT17=31                ; MASK 00020000
;       BIT18=32                ; MASK 00040000
;       BIT19=33                ; MASK 00080000
;       BIT20=34                ; MASK 00100000
;       BIT21=35                ; MASK 00200000
;       BIT22=36                ; MASK 00400000
;       BIT23=37                ; MASK 00800000
;       BIT24=38                ; MASK 01000000
;       BIT25=39                ; MASK 02000000
;       BIT26=3A                ; MASK 04000000
;       BIT27=3B                ; MASK 08000000
;       BIT28=3C                ; MASK 10000000
;       BIT29=3D                ; MASK 20000000
;       BIT30=3E                ; MASK 40000000
;       BIT31=3F                ; MASK 80000000
;
;       GPR0=40                 ; GPR 0
;       GPR1=41                 ; GPR 1
;       GPR2=42                 ; GPR 2
;       GPR3=43                 ; GPR 3
;       GPR4=44                 ; GPR 4
;       GPR5=45                 ; GPR 5
;       GPR6=46                 ; GPR 6
;       CM.SP=46                ; IN COMPATIBILITY MODE THIS IS THE STACK POINTER.
;       GPR7=47                 ; GPR 7
;       CM.PC=47                ; IN COMPATIBILITY MODE THIS IS THE PC.
;       GPR8=48                 ; GPR 8
;       GPR9=49                 ; GPR 9
;       GPR10=4A                ; GPR 10
;       GPR11=4B                ; GPR 11
;       GPR12=4C                ; GPR 12
;       AP=4C                   ; ARGUMENT POINTER
;       GPR13=4D                ; GPR 13
;       FP=4D                   ; FRAME POINTER
;       SP=4E                   ; GPR 14
;       T0=4F                   ; TEMP LOCATION 0
;
;       ZERO=50                 ; ZERO CONSTANT
;       #3=51           ; CONSTANT
;       #5=52           ; CONSTANT
;       #6=53           ; CONSTANT
;       #7=54           ; CONSTANT
;       #9=55           ; CONSTANT
;       #B=56           ; CONSTANT
;         #C=57               ; CONSTANT
;       #D=58             ; CONSTANT
;       #E=59             ; CONSTANT
;       #F=5A             ; CONSTANT
;       #13=5B            ; CONSTANT
;       #1F=5C            ; CONSTANT
;       #34=5D            ; CONSTANT
;       #3B=5E            ; CONSTANT
;       #3F=5F            ; CONSTANT
;       #4B=60            ; CONSTANT
;       #66=61            ; CONSTANT
;       #A0=62            ; CONSTANT
;       #F0=63            ; CONSTANT
;       #FF=64            ; CONSTANT
;       #1FF=65           ; CONSTANT
;       #FFF=66           ; CONSTANT
;       #2001=67                  ; CONSTANT
;       #3000=68                  ; CONSTANT
;       #7F80=69                  ; CONSTANT
;       #C000=6A                  ; CONSTANT
;       #FFE0=6B                  ; CONSTANT
;       #FFFF=6C                  ; CONSTANT
;       #1F0000=6D                ; CONSTANT
;       #200001=6E                ; CONSTANT
;       #F27000=6F                ; CONSTANT
;       #3000000=70               ; CONSTANT
;       #41F0000=71               ; CONSTANT
;       #7000000=72               ; CONSTANT
;       #3FFFFE00=73              ; CONSTANT
;       #7F800000=74              ; CONSTANT
;       #7FFFFE00=75              ; CONSTANT
;       #7FFFFF0E=76              ; CONSTANT
;       #BF800001=77              ; CONSTANT
;
;       GPR.OS=78                 ; HARDWARE INDEX TO GPRS
;       BIT.OS(3-0)=79            ; 16 LOCATION HARDWARE INDEX TO BIT MASKS
;       BIT.OS(4-0)=7B            ; 32 LOCATION HARDWARE INDEX TO BIT MASKS
;       OS=7C                     ; OS REGISTER
;       DEC.CON=7D                ; DECIMAL CARRIES
;       SIZE=7E                   ; SIZE REGISTER
;       MDT= 7E                   ; MEMORY REFERENCE DATA SIZE
;       INTERRUPT.VEC=7E          ; HARDWARE INTERRUPT VECTOR
;
;       THIS LOCATION IS THE HARDWARE CONDITION CODES.  THE CC'S CAN BE
;       READ OR ARE WRITTEN HERE.  NO OTHER BITS IN THE PSL ARE EFFECTED.
;
;       PSL.CC=7F                 ; PSL CONDITION CODES

; THESE ADDRESSES ARE ONLY ACCESSIBLE WITH THE MOV MICRO INSTRUCTION.

;       KSP=80                    ; KERNEL STACK POINTER
;       ESP=81                    ; EXECUTIVE STACK POINTER
;       SSP=82                    ; SUPERVISOR STACK POINTER
;       USP=83                    ; USER STACK POINTER
;       ISP=84                    ; INTERRUPT STACK POINTER
;       ASTLVL=85                 ; AST LEVEL
;       PCBB=86                   ; PROCESS CONTROL BLOCK BASE
;       SCBB=87                   ; SYSTEM CONTROL BLOCK BASE
;       SISR=88                     ; SOFTWARE INTERRUPT SUMMARY REGISTER
```

```
;
;       CONSOLE.COMMAND=89          ; ONE BYTE COMMAND.
;       CONSOLE.MODIFIER=8A         ; ONE BYTE MODIFIER.
;       CONSOLE.ADDRESS=8B          ; FOUR BYTES OF ADDRESS.
;       CONSOLE.DATA=8C             ; FOUR BYTES OF DATA.
;       CONSOLE.STATUS=8D           ; ONE BYTE OF STATUS.
;       PSEUDO.NICR=8E              ; COPY OF NICR THAT 8085 HAS.
;       PSEUDO.ICR=8F               ; COPY OF ICR THAT 8085 HAS.
;       DEBUG.SAVE.WR0=90           ; TEMPORARY SAVE WR0.
;       DEBUG.SAVE.WR1=91           ; TEMPORARY SAVE WR1.
;       DEBUG.SAVE.ALU.CC=92        ; TEMPORARY SAVE CONDITION CODES.
;       MEMORY.SIZE=93              ; MEMORY SIZE OF THE MACHINE.
;       CRD.LOG=94                  ; INFORMATION ABOUT LAST SOFT MEMORY ERROR.
;       PACKET.A=95                 ; TOP OF PACKET.
;       PACKET.B=96                 ; BOTTOM OF PACKET.
;       DEBUG.SAVE.ADDRESS=97       ; ADDRESS FOR EXAMINE/DEPOSIT.
;
;       SAVED.MDT=99                ; SAVED ADDRESS OF MDT FOR MMIE OPTIMIZATION.
;       P0BR=9A                     ; P0 BASE REGISTER
;       P0LR=9B                     ; P0 LENGTH REGISTER
;       P1BR=9C                     ; P1 BASE REGISTER
;       P1LR=9D                     ; P1 LENGTH REGISTER
;       SBR=9E                      ; SYSTEM BASE REGISTER
;       SLR=9F                      ; SYSTEM LENGTH REGISTER
;
;       PORT0=0A0       ; RESERVED FOR PORT.
;       PORT1=0A1       ; RESERVED FOR PORT.
;       PORT2=0A2       ; RESERVED FOR PORT.
;       PORT3=0A3       ; RESERVED FOR PORT.
;       PORT4=0A4       ; RESERVED FOR PORT.
;       PORT5=0A5       ; RESERVED FOR PORT.
;       PORT6=0A6       ; RESERVED FOR PORT.
;       PORT7=0A7       ; RESERVED FOR PORT.
;       PORT8=0A8       ; RESERVED FOR PORT.
;       PORT9=0A9       ; RESERVED FOR PORT.
;       PORT10=0AA      ; RESERVED FOR PORT.
;       PORT11=0AB      ; RESERVED FOR PORT.
;       PORT12=0AC      ; RESERVED FOR PORT.
;       PORT13=0AD      ; RESERVED FOR PORT.
;       PORT14=0AE      ; RESERVED FOR PORT.
;       PORT15=0AF      ; RESERVED FOR PORT.
;       PORT16=0B0      ; RESERVED FOR PORT.
;       PORT17=0B1      ; RESERVED FOR PORT.
;       PORT18=0B2      ; RESERVED FOR PORT.
;       PORT19=0B3      ; RESERVED FOR PORT.
;       PORT20=0B4      ; RESERVED FOR PORT.
;       PORT21=0B5      ; RESERVED FOR PORT.
;       PORT22=0B6      ; RESERVED FOR PORT.
;       PORT23=0B7      ; RESERVED FOR PORT.
;       PORT24=0B8      ; RESERVED FOR PORT.
;       PORT25=0B9      ; RESERVED FOR PORT.
;       PORT26=0BA      ; RESERVED FOR PORT.
;       PORT27=0BB      ; RESERVED FOR PORT.
;       PORT28=0BC      ; RESERVED FOR PORT.
;       PORT29=0BD      ; RESERVED FOR PORT.
;       PORT30=0BE      ; RESERVED FOR PORT.
;       PORT31=0BF      ; RESERVED FOR PORT.
```

```
;
```

```
;       XGPR0=0C0               ; BACKUP COPY OF GPR 0
;       XGPR1=0C1               ; BACKUP COPY OF GPR 1
;       XGPR2=0C2               ; BACKUP COPY OF GPR 2
;       XGPR3=0C3               ; BACKUP COPY OF GPR 3
;       XGPR4=0C4               ; BACKUP COPY OF GPR 4
;       XGPR5=0C5               ; BACKUP COPY OF GPR 5
;       XGPR6=0C6               ; BACKUP COPY OF GPR 6
;       XGPR7=0C7               ; BACKUP COPY OF GPR 7
;       XGPR8=0C8               ; BACKUP COPY OF GPR 8
;       XGPR9=0C9               ; BACKUP COPY OF GPR 9
;       XGPR10=0CA              ; BACKUP COPY OF GPR 10
;       XGPR11=0CB              ; BACKUP COPY OF GPR 11
;       XGPR12=0CC              ; BACKUP COPY OF GPR 12
;       XGPR13=0CD              ; BACKUP COPY OF GPR 13
;       XSP=0CE                 ; BACKUP COPY OF GPR 14
;
;       #14=0D0                 ; CONSTANT
;       #18=0D1                 ; CONSTANT
;       #1C=0D2                 ; CONSTANT
;       #24=0D3                 ; CONSTANT
;       #28=0D4                 ; CONSTANT
;       #2C=0D5                 ; CONSTANT
;       #2D=0D6                 ; CONSTANT
;       #30=0D7                 ; CONSTANT
;       #F8=0D8                 ; CONSTANT
;       #FC=0D9                 ; CONSTANT
;       #2D20=0DA                       ; CONSTANT
;       #140000=0DB                     ; CONSTANT
;       #150000=0DC                     ; CONSTANT
;       #160000=0DD                     ; CONSTANT
;       #170000=0DE                     ; CONSTANT
;       #180000=0DF                     ; CONSTANT
;       #1E0000=0E0                     ; CONSTANT
;       #40A10=0E1                      ; CONSTANT
;       #C0000E0=0E2                    ; CONSTANT
;       #0FFF0000=0E3                   ; CONSTANT
;       #B020FF00=0E4                   ; CONSTANT
;       #FFFFFF10=0E5                   ; CONSTANT
;       DEBUG.SAVE.T1=0E6       ; TEMPORARY SAVE T1.
;       DEBUG.SAVE.OS=0E7       ; TEMPORARY SAVE OS.
;       DEBUG.SAVE.SIZE=0E8     ; TEMPORARY SAVE SIZE VALUE.
;       SAVED.WR0=0E9           ; MM SAVED WR0
;       SAVED.WR1=0EA           ; MM SAVED WR1
;       SAVED.2ND.VAR=0EB       ; MM SAVED VAR 2
;       SAVED.ALU.CC=0EC                ; MM SAVED ALU CONDITION CODES
;       SAVED.SIZE=0ED          ; MM SAVED LS[SIZE]
;       SAVED.OS=0EE            ; MM SAVED LS[OS]
;       SAVED.REQUEST=0EF       ; MM SAVED REQUEST DATA
;       SAVED.CRD.LOG=0F0       ; MM SAVED LOCATION
;       OS.BAK=0F1              ; USED BY WARM FLOATING POINT.
;
;       THIS LOCATION IS FOR USE BY THE MFPR AND MTPR TO THE CONSOLE AND
;       TU58 CSR'S.  ALL OF THE INTERRUPT ENABLE BITS ARE HEREIN.
;
;       CONSOLE/PROGRAM I/O MODE FLAG - BIT<31> 0 - CONSOLE 1 - PROGRAM I/O MODE

;       CRD RETRY IN PROGRESS - BIT<5>  0 - NOTHING DOING 1 - IN PROGRESS
;       MACHINE CHECK IN PROGRESS - BIT<4>
;       TU58 TRANSMIT CSR IE - BIT<3>
;       TU58 RECEIVER CSR IE - BIT<2>
;       CONSOLE TRANSMIT CSR IE - BIT<1>
;       CONSOLE TRANSMIT CSR IE - BIT<0>
;
;       CONSOLE.CSR.IES=0F2       ; CONSOLE CSR IE'S AND CONSOLE/PROGRAM I/O MODE FLAG.
;       IE.TEMP1=0F3              ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION ONE
;       IE.TEMP2=0F4              ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION TWO
;       IE.TEMP3=0F5              ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION THREE
;
;       XGPR.OS=0F8               ; HARDWARE INDEX TO XGPRS
;       PORT(3-0)=0F9             ; 16 LOCATION HARDWARE INDEX TO PORT REGISTERS.
;       PORT(4-0)=0FB             ; 32 LOCATION HARDWARE INDEX TO PORT REGISTERS.
;       CCR=0FC                   ; CONSOLE READ REGISTER
;       TIMER=0FD                 ; INTERVAL TIMER VALUE.
;       ALU.CC=0FE                ; ALU CONDITION CODES
;
;       THIS LOCATION IS A WRITE ONLY REGISTER.  THE FOLLOWING BITS IN THE
;       PSL ARE AFFECTED:
;
;       COMPATIBILITY MODE - BIT<31>
;       CURRENT MODE - BITS<25:24>
;       INTERRUPT PRIORITY LEVEL (IPL) - BITS<20:16>
;       TRACE TRAP ENABLE (REALLY T OR TP) - BIT<4>
;       NEGATIVE CONDITION CODE - BIT<3>
;       ZERO CONDITION CODE - BIT<2>
;       OVERFLOW CONDITION CODE - BIT<1>
;       CARRY CONDITION CODE - BIT<0>
;
;       PSL.HW=0FF                ; HARDWARE PSL BITS


; DATA PATH CONTROL
;
; THIS FIELD IS BUILT FROM AN ADDRESS IN THE CCODE MEMORY.  THE CCODE MEMORY
; CONTAINS DATA PATH MICRO INSTRUCTIONS TO PRODUCE THE DESIRED OPERATION
; FOR THE UCODE MACROS.

; DATA PATH CONTROL FIELD FOR THE 'BASIC' MICRO INSTRUCTION
;
DP/=<21:16>,.VALIDITY=<DP.VAL>,.DEFAULT=<.SHIFT[.AND[<SDP/Q.CMP.LS>,0FF],-2]>
;
; THIS FIELD IS ONLY USED WITH THE INSTRUCTIONS WHICH CAN HAVE
; XCHG ATTACHED.
;
DP.SMALL/=<20:16>,.VALIDITY=<DP.VAL>
;
; EXCHANGE ORIGINAL WR TO LS
;
XCHG.BIT/=<21>,.DEFAULT=<0>

        YES=1
        NO=0

; DATA PATH CONTROL FIELD FOR  THE 'EXTENDED' MICRO INSTRUCTION
;
XDP/=<19:14>,.VALIDITY=<A.VAL>
```

```
; DATA PATH CONTROL FOR THE 'MOVE' MICRO INSTRUCTION
;
MDP/=<19:17>,.VALIDITY=<XD.VAL>

.PAGE
; CONDITION CODE FIELD / DATA PATH CONTEXT
;
CC/=<6:5>,.VALIDITY=<CC.VAL>,.DEFAULT=<CC/DT(LONG)&HOLD.ALU.CC>

        DT(LONG)&HOLD.ALU.CC=0          ; USE LONG CONTEXT(32 BITS) AND HOLD ALU CONDITION CODES
        DT(LONG)&SET.ALU.CC=1           ; USE LONG CONTEXT(32 BITS) AND  SET ALU CONDITION CODES
        DT(SIZE)&SET.ALU.CC=2           ; USE CONTEXT IN THE SIZE REGISTER AND SET ALU CONDITION CODES
        DT(SIZE)&MAP.ALU.CC.TO.PSL=3    ; TRANSFER ALU CONDITION CODES TO PSL CONDITION CODES HARDWARE DEPENDENT


; MEMORY DATA TYPE FIELD
;
MDT/=<6:5>,.VALIDITY=<DT.VAL>

        BYTE=0                      ; SIZE = BYTE
        WORD=1                      ; SIZE = WORD
        SIZE=2                      ; SIZE = CONTENTS OF SIZE REGISTER
        LONG=3                      ; SIZE = LONG


; SHIFT INPUT FIELD
;
; THIS CONTROL FIELD DETERMINES THE SHIFT INPUTS FOR THE FOLLOWING OPERATIONS.
; THE DIRECTION OF THE SHIFT IS DETERMINED BY THE 2901 DESTINATION CODE.
;
SI/=<13:11>,.VALIDITY=<A.VAL>

        ROT.WR=0                    ; ROTATE WR
        ROT.WR.Q=1                  ; ROTATE WR AND Q
        ASH.WR=2                    ; ARITHMETIC SHIFT OF WR
        ASH.WR.Q=3                  ; ARITHMETIC SHIFT OF WR AND Q
        MUL.SHF=4                   ; SIGNED MULTIPLY SHIFT OPERATION
        UMUL.SHF=5                  ; UNSIGNED MULTIPLY SHIFT OPERATION
        SHF.WR.Q=6                  ; LOGICAL SHIFT WR AND Q


; SKIP MICRO CONTROL FIELD
;
; THIS FIELD IS VALID FOR ALL INSTRUCTIONS EXCEPT
;
;               JUMP   MICRO INSTRUCTION
;               DECODE MICRO INSTRUCTION
;
SCTL/=<4:0>,.VALIDITY=<.AND[<SCTL.VAL>,<SKIP.PAGE.VAL>]>,.DEFAULT=<SCTL/NO.SKIP>

        N.CLR=0                     ; ALU N-BIT EQUAL ZERO
        NEQ=1                       ; ALU Z-BIT EQUAL ZERO
        BIT.SET=1                   ; ALU Z-BIT EQUAL ZERO
        V.CLR=2                     ; ALU V-BIT EQUAL ZERO
        C.SET=3                     ; ALU C-BIT EQUAL ONE
        GEQU=3                      ; ALU C-BIT EQUAL ONE
        NOT.PSL.C=4                 ; PSL C EQUAL ZERO
        BR.FALSE=5              ; BRANCH INSTRUCTION FALSE
        R.DST=6                     ; REGISTER MODE FLAG
        NO.INTERRUPT=7              ; NO INTERRUPT PENDING
        N.SET=8                     ; ALU N-BIT EQUAL ONE
        EQL=9                       ; ALU Z-BIT EQUAL ONE
        BITS.CLR=9                  ; ALU Z-BIT EQUAL ONE
        V.SET=0A                    ; ALU V-BIT EQUAL ONE
        C.CLR=0B                    ; ALU C-BIT EQUAL ZERO
        LSSU=0B                     ; ALU C-BIT EQUAL ZERO
        STATE.ZERO.SET=0C           ; STATE REGISTER BIT ZERO TRUE
        STATE.ONE.SET=0D            ; STATE REGISTER BIT ONE TRUE
        STATE.ZERO.CLR=0E           ; STATE REGISTER BIT ZERO FALSE
        STATE.ONE.CLR=0F            ; STATE REGISTER BIT ONE FALSE
        RET.NOERR=10                ; RETURN+1 IF NO MEMORY ERROR
        JMP.Z.CLR=11                ; JUMP TO (STACK) IF ALU Z = 0
        POP.USTACK=12               ; DISCARD TOP STACK ENTRY
        JMP.C.SET=13                ; JUMP TO (STACK) IF ALU C=1
        RETURN=14                   ; RETURN TO (USTACK)
        NO.SKIP=15                  ; EXECUTE NEXT INSTRUCTION
        RETURN+1=16                 ; RETURN TO (USTACK)+1
        LOOP.IF.NO.I.AND.SYNC=17;   JUMP TO (STACK) IF NO INTERRUPT AND NO ACCELERATOR SYNC.
        CONSOLE.ATTN=18             ; CONSOLE ATTENTION
        CONSOLE.ACK=19              ; CONSOLE ACKNOWLEDGE
        NO.FAST.INTERRUPT=1A        ; NO FAST INTERRUPTS PENDING
        BACKUP.MASK.VALID=1B        ; REGISTER BACKUP MASK VALID.
        LSS=1C                      ; SIGNED LESS THAN.
        MEM.REF.OK=1D               ; NO MEMORY ERROR TRUE
        SKIP=1E                     ; EXECUTE ADDRESS PLUS ONE
        SYNC=1F                     ; ACCELERATOR SYNCHRONIZATION

; JUMP MICRO CONTROL FIELD
;
; THIS FIELD IS VALID FOR THE FOLLOWING INSTRUCTIONS
;
;               JUMP   MICRO INSTRUCTION
;               DECODE MICRO INSTRUCTION
;
JCTL/=<3:0>,.VALIDITY=<.AND[<JCTL.VAL>,<JUMP.PAGE.VAL>]>,.DEFAULT=<JCTL/JUMP.VALID>

        N.CLR=0                     ; ALU N-BIT EQUAL ZERO
        NEQ=1                       ; ALU Z-BIT EQUAL ZERO
        BIT.SET=1                   ; ALU Z-BIT EQUAL ZERO
        V.CLR=2                     ; ALU V-BIT EQUAL ZERO
        C.SET=3                     ; ALU C-BIT EQUAL ONE
        GEQU=3                      ; ALU C-BIT EQUAL ONE
        JUMP=4                      ; JUMP UNCONDITIONALLY
        BR.FALSE=5                  ; BRANCH INSTRUCTION FALSE
        R.DST=6                     ; REGISTER MODE FLAG
        NO.INTERRUPT=7              ; NO INTERRUPT PENDING
        N.SET=8                     ; ALU N-BIT EQUAL ONE
        EQL=9                       ; ALU Z-BIT EQUAL ONE
        BITS.CLR=9                  ; ALU Z-BIT EQUAL ONE
        V.SET=0A                    ; ALU V-BIT EQUAL ONE
        C.CLR=0B                    ; ALU C-BIT EQUAL ZERO
        LSSU=0B                     ; ALU C-BIT EQUAL ZERO
        JSR=0C                      ; UNCONDITIONAL JUMP AND PUSH USTACK
        JSR.VALID=0D                ; JUMP AND PUSH USTACK IF IB VALID=1
        NO.JUMP.TST=0E          ; DO NOT JUMP AND SKIP IF IB VALID=0
        JUMP.VALID=0F           ; JUMP IF IB VALID=1

; JUMP ADDRESS FIELD
;
JUMP.ADRS/=<17:4>,.ADDRESS,.VALIDITY=<JUMP.VAL>

; OS ENABLE
;
OS/=<18>,.VALIDITY=<JUMP.VAL>

        NOP=0
        WITH.OS=1               ; CONCATENATE OS REGISTER TO JUMP ADDRESS


.PAGE
; BACKUP PC
;
BPC/=<18>,.VALIDITY=<DECODE.VAL>,.DEFAULT=<.CASE[<.EQL[<OPC2/>,<OPC2/DECODE>]>]OF[0,1]>

        NOP=1
        SAVE.PC=0               ; WRITE ALU DATA INTO WR[B]   (PC+1)

; DECODE ADDRESS FIELD
;
DEC.ADRS/=<17:12>,.VALIDITY=<DECODE.VAL>

; IR FUNCTION
;
IFUNC/=<11:9>,.VALIDITY=<DECODE.VAL>

        CM.EXEC=0               ; COMPATIBILITY MODE EXECUTION DISPATCH WHEN UPPER BYTE = ZERO
        SPEC=0                  ; SPECIFIER DISPATCH
        CM.IRD=1                ; COMPATIBILITY MODE INSTRUCTION DECODE DISPATCH
        FLOAT=1                 ; SPECIFIER FLOAT TYPE DISPATCH
        VAX.EXEC=2              ; VAX EXECUTION DISPATCH
        ASRC=2                  ; ADDRESS SPECIFIER DISPATCH
        VAX.IRD=3               ; VAX OPCODE DISPATCH
        VSRC=4                  ; ADDRESS SPECIFIER DISPATCH FOR BIT FIELD INSTRUCTIONS
        ESRC=5                  ; SPECIFIER DISPATCH IN INDEX MODE
        CM.DST=6                ; COMPATIBILITY MODE DESTINATION DISPATCH
        CM.SINGLE=7             ; COMPATIBILITY MODE SOP DISPATCH

; INSTRUCTION BUFFER REQUEST
;
IB.REQ/=<8>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0

        NOP=0
        IB.REQ=1                ; ENABLE HARDWARE DEPENDENT MEMORY REQUEST
;
; COMPATIBILITY MODE OPCODE SELECT
;
CM.HI/=<7>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0
```

```
        LOW.BYTE=0                ; DECODE IR BITS<7:0>
        HI.BYTE=1                 ; DECODE IR BITS <15:6>


; LOAD OS REGISTER
;
LD.OS/=<6>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0

        NOP=0
        LOAD.OS=1                 ; LOAD OS REGISTER FROM I-BUFFER


; REGISTER DESTINATION FLAG ENABLE
;
R.DST/=<5>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0

        NOP=0
        ENAB=1
;
; OPCODE SPECIFIER BIT
;
OPC.SPEC/=<4>,.VALIDITY=<DECODE.VAL>

        CM.EXEC=1                 ; COMPATIBILITY MODE EXECUTION DISPATCH
        SPEC=0                    ; SPECIFIER DISPATCH
        CM.IRD=1                  ; COMPATIBILITY MODE INSTRUCTION DECODE DISPATCH
        FLOAT=0                   ; SPECIFIER FLOAT TYPE DISPATCH
        VAX.EXEC=1                ; VAX EXECUTION DISPATCH
        ASRC=0                    ; ADDRESS SPECIFIER DISPATCH
        VAX.IRD=1                 ; VAX OPCODE DISPATCH
        VSRC=0                    ; ADDRESS SPECIFIER DISPATCH FOR BIT FIELD INSTRUCTIONS
        ESRC=0                    ; SPECIFIER DISPATCH IN INDEX MODE
        CM.DST=0                  ; COMPATIBILITY MODE DESTINATION DISPATCH
        CM.SINGLE=0               ; COMPATIBILITY MODE SOP DISPATCH
.PAGE
; MISCELLANEOUS FUNCTIONS
;
;
; THIS FIELD IS A FLAG FOR STEERING THE MICROWORD AS A MISCELLANEOUS
; INSTRUCTION OR AS A PORT INSTRUCTION.
;
MISC.PORT/=<18>

        MISC=0
        PORT=1

MISC.0/=<17>,.VALIDITY=<MISC.VAL>

        NOP=0                     ; NO OPERATION IN ALU OR CONDITION CODES.

MISC.1/=<15:12>,.VALIDITY=<MISC.VAL>

        NOP=0F                    ; NO OPERATION
        SET.CP.ATTN=0E            ; SET CPU ATTENTION FOR CONSOLE
        SET.CP.ACK=0D             ; SET CPU ACKNOWLEDGE FOR CONSOLE
        CLR.CP.ATTN.AND.ACK=0C    ; CLEAR CPU ATTENTION AND CPU ACKNOWLEDGE
        SET.HIGH.PAGE=0B          ; SET BIT FOR HIGH HALF OF WCS.
```

```
	CLR.HIGH.PAGE=0A	; CLEAR BIT FOR LOW HALF OF WCS.
```

```
        SET.PORT.I.O=9           ; SET PORT I/O MODE
        SET.ACC.I.O=8            ; SET ACCELERATOR I/O MODE
        SET.BACKUP.MASK.VALID=7 ; SET REGISTER BACKUP MASK FLAG
        SET.S1=6                 ; SET STATE ONE BIT
        SET.S0=5                 ; SET STATE ZERO BIT
        CLR.S1=4                 ; CLEAR STATE ONE BIT
        CLR.S0=3                 ; CLEAR STATE ZERO BIT
        SET.S1&CLR.S0=2          ; SET STATE ONE BIT AND CLEAR STATE ZERO BIT
        CLR.S1&SET.S0=1          ; CLEAR STATE ONE AND SET STATE ZERO
        CLR=0                    ; CLEAR STATE ONE AND STATE ZERO BITS.

MISC.2/=<11:9>,.VALIDITY=<MISC.VAL>

        NOP=0                    ; NO OPERATION
        MASK.INTERRUPTS=1        ; MASK ALL INTERRUPTS (FOR DECODE NEXT CYCLE).
        ASSERT.DA.AND.PASS.WR=2; ASSERT DATA AVAILABLE AND PASS WR[0] TO THE ACCELERATOR OR PORT.
        TRAP.ACC=3               ; TRAP ACCELERATOR
        READ.ACC.UPC=4           ; READ ACCELERATOR PROGRAM COUNTER
        TRANSFER.GRANT=5         ; RECOGNIZE PORT REQUEST.
        MASK.HALT.AND.T.BIT=6    ; MASK HALT AND T-BIT TRAPS (FOR DECODE TWO CYCLES LATER)
        WRITE.WR.TO.CONSOLE.REGISTER=7  ; SEND BITS <7:0> TO 8085.

MISC.WR/=<8:7>

MISC.WRL/=<8>
MISC.WRR/=<7>

PORT.SELECT/=<17:15>

        IDC=7

PORT.FUNC/=<14:13>

        READ=0
        WRITE=1
        CONTROL=2

R.W.FUNC/=<12:10>

        CSR=0
        DAR=1
        DATA.BYTE=2
        DATA.WORD=3
        PATTERN=4
        POSITION=5

CNTL.FUNC/=<12:10>

        CLEAR.FIFO.CNTR=0
        RESET.BR=1
        CLEAR.IDC=3
        SET.AUTO=4
        CLEAR.AUTO=5
        SEL.FIFO.A=6
        SEL.FIFO.B=7
```

```
.PAGE
```

```
; MEMORY REQUEST FIELD
;
; THIS FIELD IS USED TO INDICATE THE DESIRED OPERATION TO THE MEMORY CONTROL.
; NOTE THAT THIS IS A DUMMY FIELD. IT IS ACTUALLY MADE UP OF TWO FIELDS.
; M.FUNC1 AND M.FUNC2
;
; THE FOLLOWING SPECIAL CONSIDERATIONS MUST BE MADE:
;
; ROTATE.BYTE.RIGHT - PERFORMS 9-BIT ROTATE AND THEREFORE THE
; ANSWER MUST BE CLEANED UP WITH A ROL WR[].
;
; WORD.SWAP - PERFORMS A 15-BIT ROTATE AND THERE FORE
; THE ANSWER MUST BE CLEANED UP WITH A ROL WR[].
;
; OCTA.WRITE.P - MUST BE LONGWORD ALLIGNED.
;
; OCTA.WR.V.WCHK - MUST BE LONGWORD ALLIGNED.  THIS DATA CAN
; CROSS PAGE BOUNDARIES
;
MEM.FUNC/=<7>,.VALIDITY=0

;       PREFETCH=0                ; USED TO TAKE VAR AND ADD ONE BEFORE REFERENCE.
        WRITE.TB=01               ; WRITE TRANSLATION BUFFER
        TEST.V.RCHK=2             ; TEST WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
        READ.P=3                  ; READ WITH PHYSICAL ADDRESS
        WRITE.P=4                 ; WRITE WITH PHYSICAL ADDRESS
        TEST.V.WCHK=5             ; WRITE WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
        ROTATE.BYTE.RIGHT=6       ; ROTATE ADDRESS DATA 9 BITS RIGHT AND RETURN
        WORD.SWAP=7               ; ROTATE ADDRESS DATA 15 BITS LEFT AND RETURN
        READ.V.NOCHK=8            ; READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
        READ.V.WCHK=9             ; READ WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
        READ.V.RCHK=0A            ; READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
        READ.MAINT.VAR.INC=0B     ; TEST VAR INCREMENTING.
        WRITE.V.NOCHK=0C          ; WRITE WITH VIRTUAL ADDRESS AND NO ACCESS CHECK
        WRITE.V.WCHK=0D           ; WRITE WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
        IB.FILL=0E                ; READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK AND FILL IB
        READ.V.RCHK.IFILL=0E      ; READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK AND FILL IB
        WRITE.UBS.MAP=0F          ; WRITE TO THE UNIBUS MAP.

        OCTA.WRITE.P=10           ; WRITE OCTA DATA WITH PHYSICAL ADDRESS.
        WRITE.TB.STEP=11          ; WRITE TO TWO TB ENTRIES(WITH CORRECT ADDRESS)
        READ.UBS.MAP=12           ; READ UNIBUS MAP
        READ.TB=13                ; READ TRANSLATION BUFFER
        READ.MAINT.ADRS=14        ; RPUTS VA ON UNIBUS LINES AND READS THIS DATA
        MAINT.ECC.DATA=15         ; TEST ALL ECC FUNCTIONS.
        READ.CSR=16               ; READ CONTROL REGISTER
        READ.CNTRL.REG=16         ; READ CONTROL REGISTER
        WRITE.CNTRL.REG=17        ; WRITE CONTROL REGISTER
        WRITE.CSR=17              ; WRITE CONTROL REGISTER
        OCTA.READ.P=18            ; READ OCTA DATA WITH PHYSICAL ADDRESS.
        READ.V.WCHK.LOCKED=19     ; READ VIRTUAL ADDRESS AND WRITE ACCESS CHECK AND LOCKED
        OCTA.READ.V.RCHK=1A       ; READ OCTA WITH VIRTUAL ADDRESS AND READ ACCESS CHECK.
        READ.MAINT.BR.CHECK=1B ; TESTS BRANCHING FUNCTION.
        ISSUE.BG=1C               ; ISSUES BUS GRANT (4+ADDRESS<1:0>)
        OCTA.WR.V.WCHK=1D         ; WRITE OCTA WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK.
        QUAD.READ.V.RCHK=1E       ; READ QUAD WITH VIRTUAL ADDRESS AND READ ACCESS CHECK.
        READ.MAINT.UBS=1F         ; PUTS VA ON UNIBUS LINES AND READS THIS AS DATA.
```

```
M.FUNC2/=<18:16>,.VALIDITY=<DT.VAL>

M.FUNC1/=<8:7>,.VALIDITY=<DT.VAL>

PAR/=<23>,.DEFAULT=<.PARITY[<OPC2/>,<OS/>,<JUMP.ADRS/>,<JCTL/>]>
.UCODE
.PAGE   "MACRO DEFINITIONS                                       VER 00.02"

;       THE FORMAT FOR THIS GROUP IS TWO ADDRESS WITH A MODIFY DESTINATION.
;       THE FIRST OPERAND IS COMBINED WITH THE SECOND OPERAND AND THE
;       RESULT WRITTEN TO THE SECOND OPERAND.  CMP AND BIT INSTRUCTIONS ARE
;       READ ONLY.
;
;       DURING ALL ARITHMETIC AND LOGICAL INSTRUCTIONS THE ALU CC'S
;       CAN BE LOADED BY ADDING THE
;
;               DT(SIZE)&SET.ALU.CC
;
;       MACRO TO THE MICROWORD.  THIS SPECIFIES THAT ALU CC'S ARE LOADED
;       WITH THE 2901 CONDITIONS.  NO EXTERNAL MUNGING IS DONE.  IF THE
;       DATA OPERATION IS WITH BYTES THEN THE CC'S ARE SET
;       ACCORDING TO BITS 7-0. WORD USES BITS 15-0 AND LONGWORD USES BITS
;       31-0.  THE FOLLOWING IS A DESCRIPTION OF THE ALU CC'S VALUE
;       FOR EACH FUNCTION:
;
;       ADD -- BINARY ADDITION OF THE TWO OPERANDS
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO.
;       V - ARITHMETIC OVERFLOW
;       C - CARRY
;
;       SUB -- BINARY ADDITION OF FIRST OPERAND AND TWO'S COMPLEMENT OF THE SECOND OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO.
;       V - ARITHMETIC OVERFLOW
;       C - COMPLEMENT OF THE BORROW.
;
;       BIS -- LOGICAL OR OF THE TWO OPERANDS.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       BIC -- ONE'S COMPLEMENT OF FIRST OPERAND ANDED WITH SECOND OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       AND -- LOGICAL AND OF THE TWO OPERANDS.
;
;       N - SIGN BIT
```

```
;	Z - SET IF ANSWER IS ZERO
```

```
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       XOR -- LOGICAL EXCLUSIVE OR OF THE TWO OPERANDS.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       CMP -- PERFORM BINARY ADDITION OF FIRST OPERAND AND TWO'S COMPLEMENT OF SECOND OPERAND BUT DO NOT STORE.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO.
;       V - ARITHMETIC OVERFLOW
;       C - CARRY
;
;       BIT -- PERFORM LOGICAL AND OF THE TWO OPERANDS BUT DO NOT STORE.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       SWAP -- SWITCH THE TWO VALUES.
;
;       N - SIGN BIT OF VALUE TO WR.
;       Z - SET IF ANSWER IS ZERO OF VALUE TO WR.
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
.PAGE
;
;       MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE LS[D]
;
ADD  LS[] TO   WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP.SMALL/<.SHIFT[<.AND[<SDP/LS.PLUS.WR>,07F]>,-2]>"
SUB  LS[] FROM WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP.SMALL/<.SHIFT[<.AND[<SDP/LS.FROM.WR>,07F]>,-2]>"
BIS  LS[] TO   WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.OR.WR>,0FF]>,-2]>"
BIC  LS[] TO   WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.MASK.WR>,0FF]>,-2]>"
AND  LS[] TO   WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP.SMALL/<.SHIFT[<.AND[<SDP/LS.AND.WR>,07F]>,-2]>"
XOR  LS[] TO   WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP.SMALL/<.SHIFT[<.AND[<SDP/LS.XOR.WR>,07F]>,-2]>"

CMP  LS[] WITH WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.CMP.WR>,0FF]>,-2]>"
BIT  LS[] WITH WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.BIT.LS>,0FF]>,-2]>"
SWAP LS[] WITH WR[]             "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/SWAP.LS.WR>,0FF]>,-2]>"
;
;       THE FOLLOWING MACROS IS TO BE USED WITH ADD,SUB,
;       AND,XOR LS[] TO WR[]
;
;       IT WILL TAKE THE ORIGINAL CONTENTS OF WR[]
;       AND PUT IT IN LS[].
;
XCHG                            "XCHG.BIT/YES"
;
;       MACROS FOR THE FORM OF LS[D]<-- LS[D] OPERATE Q-REGISTER
;
ADD   Q TO   LS[]               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.PLUS.LS>,0FF]>,-2]>"
SUB   Q FROM LS[]               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.FROM.LS>,0FF]>,-2]>"
```

```
BIS    Q TO   LS[]                    "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.OR.LS>,0FF]>,-2]>"
```

```
AND   Q TO   LS[]              "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.AND.LS>,0FF]>,-2]>"
XOR   Q TO   LS[]              "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.XOR.LS>,0FF]>,-2]>"

CMP   Q WITH LS[]              "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.CMP.LS>,0FF]>,-2]>"
BIT   Q WITH LS[]              "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.BIT.Q>,0FF]>,-2]>"
;
;       MACROS FOR THE FORM OF Q-REGISTER<-- Q-REGISTER OPERATE LS[D]
;
ADD  LS[] TO   Q               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.PLUS.Q>,0FF]>,-2]>"
SUB  LS[] FROM Q               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.FROM.Q>,0FF]>,-2]>"
BIS  LS[] TO   Q               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.OR.Q>,0FF]>,-2]>"
BIC  LS[] TO   Q               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.MASK.Q>,0FF]>,-2]>"
AND  LS[] TO   Q               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.AND.Q>,0FF]>,-2]>"
XOR  LS[] TO   Q               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.XOR.Q>,0FF]>,-2]>"

CMP  LS[] WITH Q               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.CMP.Q>,0FF]>,-2]>"
BIT  LS[] WITH Q               "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.BIT.Q>,0FF]>,-2]>"
;
;       MACROS FOR THE FORM OF LS[D]<-- LS[D] OPERATE WR[B]
;
ADD  WR[] TO   LS[]            "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.LS>,0FF]>,-2]>"
SUB  WR[] FROM LS[]            "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.FROM.LS>,0FF]>,-2]>"
BIS  WR[] TO   LS[]            "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.OR.LS>,0FF]>,-2]>"
AND  WR[] TO   LS[]            "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.AND.LS>,0FF]>,-2]>"
XOR  WR[] TO   LS[]            "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.XOR.LS>,0FF]>,-2]>"

CMP  WR[] WITH LS[]            "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.CMP.LS>,0FF]>,-2]>"
BIT  WR[] WITH LS[]            "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.BIT.LS>,0FF]>,-2]>"
SWAP WR[] WITH LS[]            "SWAP LS[@2] WITH WR[@1]"
;
;       MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE WR[A]
;
ADD  WR[] TO   WR[]            "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.PLUS.WR>,03F]>"
SUB  WR[] FROM WR[]            "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.FROM.WR>,03F]>"
BIS  WR[] TO   WR[]            "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.OR.WR>,03F]>"
BIC  WR[] TO   WR[]            "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.MASK.WR>,03F]>"
AND  WR[] TO   WR[]            "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.AND.WR>,03F]>"
XOR  WR[] TO   WR[]            "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.XOR.WR>,03F]>"

CMP  WR[] WITH WR[]            "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.CMP.WR>,03F]>"
BIT  WR[] WITH WR[]            "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.BIT.WR>,03F]>"
;
;       MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE Q-REGISTER
;
ADD  Q TO   WR[]               "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.PLUS.WR>,03F]>"
SUB  Q FROM WR[]               "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.FROM.WR>,03F]>"
BIS  Q TO   WR[]               "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.OR.WR>,03F]>"
AND  Q TO   WR[]               "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.AND.WR>,03F]>"
XOR  Q TO   WR[]               "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.XOR.WR>,03F]>"

CMP  Q WITH WR[]               "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.CMP.WR>,03F]>"
BIT  Q WITH WR[]               "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/Q.BIT.WR>,03F]>"
;
;       MACROS FOR THE FORM OF Q-REGISTER<-- Q-REGISTER OPERATE WR[B]
;
ADD  WR[] TO   Q               "OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.PLUS.Q>,03F]>"
SUB  WR[] FROM Q               "OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.FROM.Q>,03F]>"
```

```
BIS  WR[] TO   Q                    "OPC1/EXTENDED,A.ADRS/@1,XDP/<.AND[<SDP/WR.OR.Q>,03F]>"
```

```
BIC  WR[] TO   Q                "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.MASK.Q>,03F]>"
AND  WR[] TO   Q                "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.AND.Q>,03F]>"
XOR  WR[] TO   Q                "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.XOR.Q>,03F]>"

CMP  WR[] WITH Q                "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.CMP.Q>,03F]>"
BIT  WR[] WITH Q                "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/Q.BIT.WR>,03F]>"
.PAGE
;
;       THE FORMAT OF THIS GROUP IS THREE ADDRESS TYPE. THE FIRST AND SECOND
;       OPERANDS ARE READ AND THE RESULT IS WRITTEN INTO THE THIRD OPERAND.
;
;       MACROS FOR THE FORM OF Q-REGISTER<-- WR[B] OPERATE WR[A]
;
ADD  WR[] PLUS WR[] TO Q        "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.PLUS.WR.Q>,03F]>"
SUB  WR[] FROM WR[] TO Q        "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.FROM.WR.Q>,03F]>"
BIS  WR[] WITH WR[] TO Q        "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.OR.WR.Q>,03F]>"
BIC  WR[] WITH WR[] TO Q        "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.MASK.WR.Q>,03F]>"
AND  WR[] WITH WR[] TO Q        "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.AND.WR.Q>,03F]>"
XOR  WR[] WITH WR[] TO Q        "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.XOR.WR.Q>,03F]>"
;
;       MACROS FOR THE FORM OF Q-REGISTER<-- WR[B] OPERATE LS[D]
;
ADD  WR[] PLUS LS[] TO Q        "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.LS.Q>,0FF]>,-2]>"
SUB  WR[] FROM LS[] TO Q        "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.FROM.LS.Q>,0FF]>,-2]>"
BIS  WR[] WITH LS[] TO Q        "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.OR.LS.Q>,0FF]>,-2]>"
AND  WR[] WITH LS[] TO Q        "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.AND.LS.Q>,0FF]>,-2]>"
XOR  WR[] WITH LS[] TO Q        "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.XOR.LS.Q>,0FF]>,-2]>"
;
;       MACROS FOR THE FORM OF Q-REGISTER <-- LS[D] OPERATE WR[B]
;
ADD LS[] PLUS WR[] TO Q "ADD WR[a2] PLUS LS[a1] TO Q"
SUB  LS[] FROM WR[] TO Q        "OPC/BASIC,B.ADRS/a2,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR.Q>,0FF]>,-2]>"
BIS LS[] WITH WR[] TO Q "BIS WR[a2] WITH LS[a1] TO Q"
AND LS[] WITH WR[] TO Q         "AND WR[a2] WITH LS[a1] TO Q"
XOR LS[] WITH WR[] TO Q         "XOR WR[a2] WITH LS[a1] TO Q"


.PAGE
; SPECIAL FUNCTION ADD/SUB OPERATIONS

ADD (WR[] TO WR[])+1            "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.PLUS.WR+1>,03F]>"
ADD (LS[] TO WR[])+1            "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.PLUS.WR+1>,0FF]>,-2]>"
ADD (WR[] TO LS[])+1            "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.LS+1>,0FF]>,-2]>"

ADD OS PLUS WR[] TO LS[]        "OPC1/MOVE,B.ADRS/a1,XD.ADRS/a2,MDP/<.SHIFT[<.AND[<SDP/WR.PLUS.OS>,3F]>,-3]>"

SUB (WR[] FROM WR[])-1          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.FROM.WR-1>,03F]>"
SUB (LS[] FROM WR[])-1          "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR-1>,0FF]>,-2]>"
SUB (WR[] FROM LS[])-1          "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.FROM.LS-1>,0FF]>,-2]>"


SUB  WRB[] FROM WRA[] TO WRB[]  "OPC1/EXTENDED,B.ADRS/a1,A.ADRS/a2,XDP/<.AND[<SDP/WR.FROM.WR.WR>,03F]>"
SUB  LS[]  FROM WR[]  TO LS     "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR.LS>,0FF]>,-2]>"
SUB  WR[] FROM LS[]  TO WR      "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WRA.FROM.LS.WRB>,0FF]>,-2]>"

SUB  WRA[] FROM Q    TO WRB[]   "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.FROM.Q.WR>,03F]>"
SUB  LS[]  FROM Q    TO LS[]    "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.FROM.Q.LS>,0FF]>,-2]>"
SUB  Q     FROM WR[] TO Q        "OPC1/EXTENDED,B.ADRS/@1,XDP/<.AND[<SDP/Q.FROM.WR.Q>,03F]>"
SUB  Q     FROM LS[] TO Q         "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.FROM.LS.Q>,0FF]>,-2]>"

ADD Q PLUS WR[] TO LS[]           "OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.PLUS.WR.TO.LS>,0FF]>,-2]>"
SUB Q FROM WR[] TO LS[]           "OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.FROM.WR.TO.LS>,0FF]>,-2]>"
ADD Q PLUS LS[] TO WR[] "OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/Q.PLUS.LS.TO.WR>,0FF]>,-2]>"

ADD Q TO WR[] XCHG TO LS[]        "OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.Q.XCHG>,0FF]>,-2]>"

.PAGE
; MOVE MACRO INSTRUCTIONS
;
;       DURING THE MOVE INSTRUCTIONS THE ALU CC'S CAN BE LOADED BY
;       ADDING THE
;
;               DT(SIZE)&SET.ALU.CC
;
;       MACRO TO THE MICROWORD.  THE FOLLOWING IS A DESCRIPTION OF THE ALU
;       CC'S VALUE FOR EACH FUNCTION:
;
;       MOV -- PUT FIRST OPERAND ON TOP OF THE SECOND OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       MCOM -- PUT ONE'S COMPLEMENT OF THE FIRST OPERAND ON TOP OF THE SECOND OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       MNEG -- PUT TWO'S COMPLEMENT OF THE FIRST OPERAND ON TOP OF THE SECOND OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO.
;       V - ARITHMETIC OVERFLOW
;       C - CARRY
MOV Q    TO LS[]                  "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.Q.LS>,0FF]>,-2]>"
MOV Q    TO WR[]                  "OPC1/EXTENDED,B.ADRS/@1,XDP/<.AND[<SDP/MOV.Q.WR>,03F]>"
MOV LS[] TO Q                     "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.LS.Q>,0FF]>,-2]>"
MOV Q TO WR[] XCHG TO LS[]        "OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.Q.WR&WR.LS>,0FF]>,-2]>"

MOV IB.DATA TO OS                 "OPC2/DECODE,IFUNC/SPEC,R.DST/NOP,IB.REQ/IB.REQ,JCTL/NO.JUMP.TST,LD.OS/LOAD.OS"
MOV CM.IB.DATA TO OS              "OPC2/DECODE,IFUNC/SPEC,R.DST/NOP,IB.REQ/NOP,JCTL/NO.JUMP.TST,LD.OS/LOAD.OS"

MOV MEM.DATA TO WR[]              "OPC1/MOVE,B.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.WR>,3F]>,-3]>,XD.ADRS/WR.BACKUP"
MOV MEM.DATA TO WR[] XCHG TO LS[]       "OPC1/MOVE,B.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.WR>,3F]>,-3]>,XD.ADRS/@2"
MOV MEM.DATA TO LS[]              "OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.LS>,3F]>,-3]>"
MOV LS[]     TO WR[]              "OPC1/MOVE,XD.ADRS/@1,B.ADRS/@2,MDP/<.SHIFT[<.AND[<SDP/MOV.LS.WR>,3F]>,-3]>"
MOV WR[]     TO LS[]              "OPC1/MOVE,B.ADRS/@1,XD.ADRS/@2,MDP/<.SHIFT[<.AND[<SDP/MOV.WR.LS>,3F]>,-3]>"
MOV WR[]     TO WR[]              "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.PLUS.0.TO.WR>,03F]>"
MOV WR[]     TO Q                 "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.OR.WR.Q>,03F]>"
MOV XWR[]    TO  Q                "OPC1/MOVE,XB.ADRS/@1,XD.ADRS/0,MDP/<.SHIFT[<.AND[<SDP/MOV.XWR.Q>,3F]>,-3]>"

MOV FPA TO LS[]                   "OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.ACC.LS>,3F]>,-3]>"
MOV PORT TO LS[]                    "OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.ACC.LS>,3F]>,-3]>,CC/DT(LONG)&HOLD.ALU.CC"
```

```
MCOM WR[] TO LS[]               "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.COM.LS>,0FF]>,-2]>"
MCOM LS[] TO WR[]               "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.COM.WR>,0FF]>,-2]>"
MNEG WR[] TO LS[]               "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.NEG.LS>,0FF]>,-2]>"
MNEG LS[] TO WR[]               "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.NEG.WR>,0FF]>,-2]>"
MNEG WR[] TO WR[]               "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.NEG.WR>,03F]>"
MCOM WR[] TO WR[]       "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.COM.WR>,03F]>"
MINC WR[] TO WR[]       "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.INC.WR>,03F]>"
MDEC WR[] TO WR[]       "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.DEC.WR>,03F]>"
MNEG Q TO LS[]          "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.NEG.LS>,0FF]>,-2]>"
.PAGE
;
; SINGLE OPERAND OPERATIONS
;
;       THIS FORM OF INSTRUCTION PERFORMS THE DESIRED OPERATION ON THE OPERAND
;       SPECIFIED.
;
;               CLEAR
;               NEGATE
;               INCREMENT
;               DECREMENT
;               COMPLEMENT
;               TEST
;
;       DURING THE SINGLE OPERAND INSTRUCTIONS THE ALU CC'S CAN BE
;       LOADING USING THE
;
;               DT(SIZE)&SET.ALU.CC
;
;       MACRO IN THE MICROWORD.  THE FOLLOWING IS A DESCRIPTION OF THE
;       ALU CC'S FOR EACH FUNCTION:
;
;       CLR -- STORE A ZERO IN THE OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       NEG -- PERFORM TWO'S COMPLEMENT OF THE OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO.
;       V - ARITHMETIC OVERFLOW
;       C - CARRY
;
;       INC -- ADD ONE TO THE OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO.
;       V - ARITHMETIC OVERFLOW
;       C - CARRY
;
;       DEC -- SUBTRACT ONE FROM THE OPERAND.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO.
```

```
;	V - ARITHMETIC OVERFLOW
```

```
;       C - CARRY
;
;       COM -- PERFORM ONE'S COMPLEMENT OF THE OPERAND (XNOR WITH ZERO).
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO.
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       TST -- ADD ZERO TO THE OPERAND TO SET THE CONDITION CODES.
;
;       N - SIGN BIT
;       Z - SET IF ANSWER IS ZERO
;       V - ARITHMETIC OVERFLOW (IN THIS CASE ALWAYS ZERO)
;       C - CARRY (IN THIS CASE ZERO)
;
CLR Q                           "OPC1/EXTENDED,A.ADRS/0,B.ADRS/0,XDP/<.AND[<SDP/WR.XOR.WR.Q>,03F]>"
CLR LS[]                        "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/CLR.LS>,0FF]>,-2]>"
CLR WR[]                        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.XOR.WR>,03F]>"

NEG  Q                          "OPC1/EXTENDED,XDP/<.AND[<SDP/NEG.Q>,03F]>"
NEG  LS[]                       "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/NEG.LS>,0FF]>,-2]>"
NEG  WR[]                       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.NEG.WR>,03F]>"

INC  Q                          "OPC1/EXTENDED,XDP/<.AND[<SDP/INC.Q>,03F]>"
INC  LS[]                       "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/INC.LS>,0FF]>,-2]>"
INC  WR[]                       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.INC.WR>,03F]>"

DEC  Q                          "OPC1/EXTENDED,XDP/<.AND[<SDP/DEC.Q>,03F]>"
DEC  LS[]                       "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/DEC.LS>,0FF]>,-2]>"
DEC  WR[]                       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.DEC.WR>,03F]>"

COM  Q                          "OPC1/EXTENDED,XDP/<.AND[<SDP/COM.Q>,03F]>"
COM  LS[]                       "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/COM.LS>,0FF]>,-2]>"
COM  WR[]                       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.COM.WR>,03F]>"

TST WR[]                        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.PLUS.0.TO.WR>,03F]>"
TST Q                           "OPC1/EXTENDED,XDP/<.AND[<SDP/Q.PLUS.0>,03F]>"

NOP                             "OPC/BASIC,D.ADRS/0,B.ADRS/0,DP/<.SHIFT[<.AND[<SDP/Q.CMP.LS>,0FF]>,-2]>"
.PAGE
;
; MACROS FOR SHIFT OPERATION ON WR[B] AND/OR Q-REGISTER
;
;       DURING THE SHIFT OPERATIONS THE ALU CC'S CAN BE LOADED BY
;       USING THE
;
;               DT(SIZE)&SET.ALU.CC
;
;       MACRO WITH THE MICROWORD.  THE FOLLOWING IS A DESCRIPTION OF THE
;       ALU CC'S WITH EACH FUNCTION.
;
;       ROR WR[] -- ROTATE 32 BIT VALUE ONE POSITION RIGHT.
;
;       N - SIGN OF INPUT
;       Z - SET IF INPUT IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       ROL WR[] -- ROTATE 32 BIT VALUE ONE POSITION LEFT.
;
;       N - SIGN OF INPUT
;       Z - SET IF INPUT IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       ASHR WR[] -- SHIFT 32 BIT VALUE RIGHT ONE POSITION AND SIGN EXTEND.
;
;       N - SIGN OF INPUT
;       Z - SET IF INPUT IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       ASHL WR[] -- SHIFT 32 BIT VALUE LEFT ONE POSITION AND INSERT A ZERO IN BOTTOM BIT.
;
;       N - SIGN OF INPUT
;       Z - SET IF INPUT IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       ASHRC WR[].Q -- SHIFT 64 BIT WR.Q ONE POSITION LEFT AND SIGN EXTEND.
;
;       N - SIGN OF INPUT WR[]
;       Z - SET IF INPUT WR[] IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       RORC WR[].Q -- ROTATE 64 BIT WR.Q ONE POSITION RIGHT.
;
;       N - SIGN OF INPUT WR[]
;       Z - SET IF INPUT WR[] IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       ASHLC WR[].Q -- SHIFT 64 BIT WR.Q ONE POSITION LEFT AND INSERT ZERO.
;
;       N - SIGN OF INPUT WR[]
;       Z - SET IF INPUT WR[] IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       ROLC WR[].Q -- ROTATE 64 BIT WR.Q ONE POSITION LEFT.
;
;       N - SIGN OF INPUT WR[]
;       Z - SET IF INPUT WR[] IS ZERO
;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;
;       SHL2 WR[] -- SHIFT 32 BIT VALUE TWO POSITIONS LEFT INSERTING ZEROES IN THE BOTTOM BITS.
;
;       N - SIGN OF INPUT WR[]+WR[]
;       Z - SET IF INPUT WR[]+WR[] IS ZERO
;       V - OVERFLOW OF INPUT WR[]+WR[]
;       C - CARRY OF INPUT WR[]+WR[]
```

```
;
```

```
ROR  WR[]                        "OPC1/EXTENDED,B.ADRS/a1,SI/ROT.WR,XDP/<.AND[<SDP/ASHR>,03F]>"

ROL  WR[]                        "OPC1/EXTENDED,B.ADRS/a1,SI/ROT.WR,XDP/<.AND[<SDP/ASHL>,03F]>"


ASHR  WR[]                       "OPC1/EXTENDED,B.ADRS/a1,SI/ASH.WR,XDP/<.AND[<SDP/ASHR>,03F]>"

ASHL  WR[]                       "OPC1/EXTENDED,B.ADRS/a1,SI/ASH.WR,XDP/<.AND[<SDP/ASHL>,03F]>"

ASHRC  WR[].Q                    "OPC1/EXTENDED,B.ADRS/a1,SI/ASH.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>"

RORC WR[].Q                      "OPC1/EXTENDED,B.ADRS/a1,SI/ROT.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>"

ASHLC  WR[].Q                    "OPC1/EXTENDED,B.ADRS/a1,SI/ASH.WR.Q,XDP/<.AND[<SDP/ASHLC>,03F]>"

ROLC WR[].Q                      "OPC1/EXTENDED,B.ADRS/a1,SI/ROT.WR.Q,XDP/<.AND[<SDP/ASHLC>,03F]>"

SHL2  WR[]                       "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a1,SI/2,XDP/<.AND[<SDP/SHL2>,03F]>"

COND.ADD&SHIFT  WR[] TO WR[].Q   "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,SI/MUL.SHF,XDP/<.AND[<SDP/COND.ADD&SHIFT>,03F]>"

COND.SUB&SHIFT  WR[] FROM WR[].Q "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,SI/MUL.SHF,XDP/<.AND[<SDP/COND.SUB&SHIFT>,03F]>"

UNSIGNED.MUL WR[] TO WR[].Q      "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,SI/UMUL.SHF,XDP/<.AND[<SDP/COND.ADD&SHIFT>,03F]>"
SHR WR[]                         "OPC1/EXTENDED,B.ADRS/a1,SI/SHF.WR.Q,XDP/<.AND[<SDP/ASHR>,03F]>"
SHL WR[]                         "ASHL WR[a1]"
SHRC WR[].Q                      "OPC1/EXTENDED,B.ADRS/a1,SI/SHF.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>"
SHLC WR[].Q                      "ASHLC WR[].Q"


; MEMORY REQUEST MACRO

; *     THIS MACRO IS NOT FOR GENERAL USE!!!                *
; *     THIS IS ONLY FOR USE IN THE FOLLOWING MACRO         *
MEM.FUNC[]                       "M.FUNC2/<.SHIFT[<MEM.FUNC/a1>,-2]>,M.FUNC1/<.AND[<MEM.FUNC/a1>,3]>"

MEM.REQ[] ADRS[] DT[]            "OPC2/MEM.REQ,MEM.FUNC[a1],D.ADRS/a2,MDT/a3"
WRITE.MEM LS[]                   "OPC1/MOVE,XD.ADRS/a1,MDP/<.SHIFT[<.AND[<SDP/MOV.LS.MEM>,3F]>,-3]>"
.PAGE
;
; MACROS FOR MISCELLANEOUS OPERATIONS
;
;       IN THE NEBULA MICROPROGRAMMING LANGUAGE THERE IS A NEED FOR
;       VARIOUS UNIQUE FUNCTIONAL OPERATIONS.  THESE ARE FOUND IN THE
;       MISC INSTRUCTIONS.  MOST FUNCTIONS ARE EXECUTED BY INSERTING
;       THE REQUEST INSIDE THE MISC[] AND MISC2[] MACROS.
;
MISC []                          "OPC2/MISC,MISC.1/a1,MISC.2/NOP,MISC.PORT/MISC"
MISC2 []                         "OPC2/MISC,MISC.1/NOP,MISC.2/a1,MISC.PORT/MISC"
MISC [] WR[]                     "MISC [a1],MISC.WRL/0,MISC.WRR/a2"
MISC2 [] WR[]                    "MISC2 [a1],MISC.WRL/0,MISC.WRR/a2"

PORT.CONTROL []                  "OPC2/MISC,MISC.PORT/PORT,CNTL.FUNC/a1,PORT.SELECT/IDC,PORT.FUNC/CONTROL"
PORT.WRITE PORT.ADRS[] WITH WR[]        "OPC2/MISC,MISC.PORT/PORT,R.W.FUNC/a1,PORT.SELECT/IDC,PORT.FUNC/WRITE,MISC.WRL/0,MISC.WRR/ -
cont> a2"
PORT.READ PORT.ADRS[]   "OPC2/MISC,MISC.PORT/PORT,R.W.FUNC/a1,PORT.SELECT/IDC,PORT.FUNC/READ"
;
```

```
;       A FEW CASES OF THE MISCELLANEOUS FUNCTIONS HAVE BEEN CALLED OUT AS
;       UNIQUE FUNCTIONS IN THE MICROPROGRAMMING LANGUAGE SET.  THESE
;       ARE LISTED BELOW.
;
;       THE FOLLOWING TO MACROS SEND DATA TO EXTERNAL ACCELLERATORS.
;
ASSERT.DA.AND.PASS.WR0          "MISC2 [ASSERT.DA.AND.PASS.WR] WR[0]"
ASSERT.DA.AND.PASS.WR1          "MISC2 [ASSERT.DA.AND.PASS.WR] WR[1]"
;
;       STATE REGISTER MACROS - THE POSSIBLE CHANGES TO THE STATE
;       BITS ARE SPECIFIED AS INDIVIDUAL FUNCTIONS.
;
CLR.STATE.ZERO                  "OPC2/MISC,MISC.1/CLR.S0,MISC.2/NOP,MISC.PORT/MISC"
CLR.STATE.ONE                   "OPC2/MISC,MISC.1/CLR.S1,MISC.2/NOP,MISC.PORT/MISC"
SET.STATE.ZERO                  "OPC2/MISC,MISC.1/SET.S0,MISC.2/NOP,MISC.PORT/MISC"
SET.STATE.ONE                   "OPC2/MISC,MISC.1/SET.S1,MISC.2/NOP,MISC.PORT/MISC"
SET.STATE.ZERO&CLR.STATE.ONE    "OPC2/MISC,MISC.1/CLR.S1&SET.S0,MISC.2/NOP,MISC.PORT/MISC"
SET.STATE.ONE&CLR.STATE.ZERO    "OPC2/MISC,MISC.1/SET.S1&CLR.S0,MISC.2/NOP,MISC.PORT/MISC"
CLR.STATE.ONE&CLR.STATE.ZERO    "OPC2/MISC,MISC.1/CLR,MISC.2/NOP,MISC.PORT/MISC"
SET.BACKUP.MASK.VALID           "OPC2/MISC,MISC.1/SET.BACKUP.MASK.VALID,MISC.2/NOP,MISC.PORT/MISC"


; CONTEXT SIZE AND CONDITION CODE MACROS

DT(SIZE)&MAP.ALU.CC.TO.PSL      "CC/DT(SIZE)&MAP.ALU.CC.TO.PSL"
DT(LONG)&SET.ALU.CC             "CC/DT(LONG)&SET.ALU.CC"
DT(SIZE)&SET.ALU.CC             "CC/DT(SIZE)&SET.ALU.CC"

.PAGE
; JUMP AND MICRO SEQUENCER CONTROL MACROS
;
;       THE FOLLOWING MACROS ARE USED TO CONTROL THE PROGRAMS FLOW OF
;       CONTROL.
;
;       JMP [] -- TRANSFER TO NEW ADDRESS UNCONDITIONALLY.
;
;       JMP.OS [] -- TRANSFER TO RESULTANT ADDRESS OF LOGICAL OR OF ADDRESS
;               IN THE INSTRUCTION AND THE OS REGISTER UNCONDITIONALLY.
;
;       JMP.IF[] TO [] -- TRANSFER TO NEW ADDRESS ONLY IF JUMP
;               CONDITION IS MET.
;
;       JSR [] -- TRANSFER TO NEW ADDRESS UNCONDITIONALLY AND PUT
;               RETURN ADDRESS ON THE SEQUENCER STACK.
;
;       JSR.OS [] -- TRANSFER TO RESULTANT ADDRESS OF LOGICAL OR OF
;               ADDRESS IN THE INSTRUCTION AND THE OS REGISTER.
;               THE RETURN ADDRESS IS ALSO PUSHED ON THE SEQUENCER STACK.
;
;       RETURN -- TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER STACK AND
;                POP THE STACK.
;
;       RETURN+1 -- TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER
;               STACK PLUS ONE AND POP THE STACK.
;
;       RETURN+1.IF(MEM.REF.OK) -- IF THE MOST RECENT MEMORY REQUEST HAD
;               NO ERROR THEN TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER
;               STACK PLUS ONE AND POP THE STACK.  IF THERE WAS AN
```

```
;		ERROR THEN SKIP OVER THE ENEXT INSTRUCTION.
```

```
;
;       LOOP.IF(NEQ) -- IF THE ALU Z-BIT IS ZERO (LAST VALUE CALCULATED
;               WHEN THE ALU.CC'S WERE SET DID NOT RESULT AS ZERO)
;               THEN TRANSFER TO THE ADDRESS ON TOP OF THE SEQUENCER
;               STACK.  THE STACK IS NOT POPPED.  IF THE ALU Z-BIT
;               IS ONE THEN CONTINUE WITH THE NEXT INSTRUCTION.
;               (UPC+1) AND THE STACK IS POPPED.
;
;       LOOP.IF(GEQU) -- IF THE ALU C-BIT IS ONE (LAST VALUE CALCULATED WHEN
;               THE ALU.CC'S WERE SET WAS GREATER THAN OR EQUAL TO ZERO)
;               THEN TRANSFER TO THE ADDRESS ON TOP OF THE SEQUENCER
;               STACK.  THE STACK IS NOT POPPED.  IF THE ALU C-BIT
;               IS ZERO THEN CONTINUE WITH THE NEXT INSTRUCTION
;               (UPC+1) AND THE STACK IS POPPED.
;
;       LOOP.IF(NO INTERRUPT AND NO SYNC) -- IF THE INTERRUPT CONDITION
;               IS NOT TRUE AND THE ACCELERATOR SYNC CONDITION IS NOT
;               TRUE THEN TRANSFER TO THE ADDRESS ON THE TOP OF THE SEQUENCER
;               STACK.  THE STACK IS NOT POPPED.  IF THE INTERRUPT
;               CONDITION IS TRUE THEN CONTINUE WITH THE NEXT INSTRUCTION
;               (UPC+1) AND THE SEQUENCER STACK IS NOT POPPED.  IF THE ACCELERATOR
;               SYNC IS TRUE THEN CONTINUE WITH THE NEXT INSTRUCTION
;               (UPC+1) AND THE SEQUENCER STACK IS POPPED.
;
;       SKIP.IF[] -- IF THE SKIP CONDITION IS SATISFIED THEN TRANSFER
;               TO UPC+2 ELSE TRANSFER TO UPC+1.
;

JMP []                          "OPC2/JUMP,JUMP.ADRS/@1,JCTL/JUMP"
JMP.OS []                       "OPC2/JUMP,JUMP.ADRS/@1,OS/WITH.OS,JCTL/JUMP"
JMP.IF[] TO []                  "OPC2/JUMP,JCTL/@1,JUMP.ADRS/@2"
JSR []                          "OPC2/JUMP,JCTL/JSR,JUMP.ADRS/@1"
JSR.OS []                       "OPC2/JUMP,JCTL/JSR,JUMP.ADRS/@1,OS/WITH.OS"

RETURN                          "SCTL/RETURN"
RETURN+1                        "SCTL/RETURN+1"
RETURN+1.IF(MEM.REF.OK)         "SCTL/RET.NOERR"

LOOP.IF(NEQ)                    "SCTL/JMP.Z.CLR"
LOOP.IF(GEQU)                   "SCTL/JMP.C.SET"
LOOP.IF(NO INTERRUPT AND NO SYNC) ELSE SKIP.IF(SYNC)    "SCTL/LOOP.IF.NO.I.AND.SYNC"

SKIP.IF[]                       "SCTL/@1"
SKIP                            "SCTL/SKIP"
.PAGE
;
; INSTRUCTION STREAM MACROS
;
; *     THIS MACRO IS NOT FOR GENERAL USE!!!            *
; *     THIS IS ONLY FOR USE IN THE FOLLOWING MACROS    *
DEC[]                           "OPC2/DECODE,DEC.ADRS/<.SHIFT[<JUMP.ADRS/@1>,-8]>"

DECODE.SPEC ADRS[]              "DEC[@1],IFUNC/SPEC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/SPEC"
DECODE.ASRC ADRS[]              "DEC[@1],IFUNC/ASRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/ASRC"
DECODE.ESRC ADRS[]              "DEC[@1],IFUNC/ESRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/ESRC"
DECODE.VSRC ADRS[]              "DEC[@1],IFUNC/VSRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/VSRC"
DECODE.FLOAT ADRS[]             "DEC[@1],IFUNC/FLOAT,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/FLOAT"
DECODE.OPC1 ADRS[]                 "DEC[@1],IFUNC/VAX.IRD,R.DST/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,OPC.SPEC/VAX.IRD"
```

```
EXEC.DISPATCH ADRS[]            "DEC[a1],IFUNC/VAX.EXEC,IB.REQ/NOP,R.DST/NOP,CM.HI/LOW.BYTE,JCTL/JSR,OPC.SPEC/VAX.EXEC"

DECODE.CM.OPC ADRS[]            "DEC[a1],IFUNC/CM.IRD,CM.HI/HI.BYTE,OPC.SPEC/CM.IRD,LD.OS/LOAD.OS"
DECODE.CM.DST ADRS[]            "DEC[a1],IFUNC/CM.DST,OPC.SPEC/CM.DST,LD.OS/LOAD.OS,R.DST/ENAB"
DECODE.CM.SINGLE ADRS[]         "DEC[a1],IFUNC/CM.SINGLE,OPC.SPEC/CM.SINGLE"
CM.EXEC ADRS[]                  "DEC[a1],IFUNC/CM.EXEC,JCTL/JUMP,OPC.SPEC/CM.EXEC,LD.OS/LOAD.OS"

SAVE.PC WR0                     "BPC/SAVE.PC"
SAVE.PC WR4                     "BPC/SAVE.PC"
SAVE.CM.PC WR0                  "BPC/SAVE.PC"
SAVE.CM.PC WR4                  "BPC/SAVE.PC"


; THESE SKIP CONDITIONS ARE USED FOR THE DECODE MICRO INSTRUCTION
;
SKIP.IF(IB VALID)               "JCTL/NO.JUMP.TST"
JMP.IF(IB VALID)                "JCTL/JUMP.VALID"
JSR.IF(IB VALID)                "JCTL/JSR.VALID"
;
;       THESE SKIP CONDITIONS ARE TO BE USED WITH THE COMPATIBILITY MODE DECODE
;       MACROS.
;
JMP.TO []                       "JCTL/JUMP,DEC[a1]"
JSR.TO []                       "JCTL/JSR,DEC[a1]"
.BIN
.PAGE
.TOC    "FIELD DEFINTITIONS FOR DATA PATH CONTROL MICRO WORD    VER 00.01"

; THIS SECTION IS FOR THE DATA PATH CONTROL ROMS
;

.CCODE
SDP/=<8:0>,.ADDRESS,.VALIDITY=0
;
; THIS FIELD CORRESPONDS TO THE 2901 ALU FUNCTION CONTROL PINS ... I5,I4,I3
;
ALU/=<2:0>,.DEFAULT=<ALU/R.OR.S>

        R.PLUS.S=0              ; ADD R WITH S
        S.MINUS.R=1             ; SUBTRACT R FROM S
        R.MINUS.S=2             ; SUBTRACT S FROM R
        R.OR.S=3                ; OR R WITH S
        R.AND.S=4               ; AND R WITH S
        NOTR.AND.S=5            ; COMPLEMENT R THEN AND WITH S
        R.XOR.S=6               ; XOR R WITH S
        R.XNOR.S=7              ; XOR R WITH S THEN COMPLEMENT THE RESULT
SRC/=<8:6>,.DEFAULT=<SRC/D.0>

        0.Q=2                   ; RMX=0         SMX=Q
        0.B=3                   ; RMX=0         SMX=B
        A.Q=0                   ; RMX=A         SMX=Q
        A.B=01                  ; RMX=A         SMX=B
        D.Q=06                  ; RMX=D         SMX=Q
        D.0=7                   ; RMX=D         SMX=0
        0.A=4                   ; RMX=0         SMX=A
        D.A=5                   ; RMX=D         SMX=A
;
```

```
; THIS FIELD CORRESPONDES TO THE 2901 ALU DESTINATION
; CONTROL PINS I8, I7 AND I6.
;
DST/=<5:3>,.DEFAULT=<DST/NOP>

        LOAD.Q=0                ; LOAD Q-REGISTER
        NOP=1                   ; HOLD ALL REGISTERS
        WRITE.B.A=2             ; WRITE RAM B-PORT AND OUTPUT RAM A-PORT
        WRITE.B.F=3             ; WRITE RAM B-PORT AND OUTPUT ALU
        RSHF.RAM.Q=4            ; RIGHT SHIFT RAM AND Q
        RSHF.RAM=5              ; RIGHT SHIFT RAM
        LSHF.RAM.Q=6            ; LEFT SHIFT RAM AND Q
        LSHF.RAM=7              ; LEFT SHIFT RAM

CIN/=<9>,.DEFAULT=0

        NO.CIN=0                ; NO CARRY IN TO ALU
        CIN=1                   ; CARRY IN TO ALU



.PAGE
.TOC    "CONTROL ROM CONTENTS                               VER 00.01"
.SEQUENTIAL
; THESE ARE THE DATA PATH CONTROL ROM CONTENTS
;
; THE FOLLOWING LOCATIONS ARE USED BY THE DECODE.IS MICRO INSTRUCTION
;
; THIS INSTRUCTION WILL DO PC+1 OPERATIONS
;
;               BACK UP PC IN 2901 RAM
000:
DECODE.IS:
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
;
;               DON'T BACK UP PC
;
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
```

```
SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
```

```
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
.PAGE
;  THESE ADDRESSES ARE USED BY THE 'JUMP' MICRO INSTRUCTION
;
; THE FOLLOWING CONTROL CODES WILL INSURE THAT NO DATA IS DESTROYED WITHIN
; THE 2901 DURING JUMP OR JSR OPERATIONS.
;
020:
JUMP:
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
.PAGE
; THESE LOCATIONS ARE USED BY THE MISC' MICRO INSTRUCTION
;
; THE FOLLOWING CONTROL CODES WILL INSURE THAT NO DATA IS DESTROYED WITHIN
; THE 2901 DURING MISC INSTRUCTION EXECUTION.
;
040:
MISC:
```

```
SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
```

```
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
.PAGE
; THESE ADDRESSES ARE USED BY THE MEM.REQ MICRO INSTRUCTION
;
; THIS INSTRUCTION CAUSES WR[5] TO BE LOADED WITH THE VIRTUAL ADDRESS
; OF THE MEMORY REFERENCE.
060:
MEM.REQ:
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
```

```
SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
```

```
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
.PAGE
; THESE ARE THE ADDRESSES FOR THE EXTENDED MICRO INSTRUCTION
; ADDRESSES START AT 80-FF
080:

WR.PLUS.0.TO.WR:
        SRC/0.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
WR.INC.WR:
        SRC/0.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
WR.NEG.WR:
        SRC/0.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
WR.COM.WR:
        SRC/0.A,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN

WR.DEC.WR:
        SRC/0.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
FILL85:
        SRC/0.A
MOV.Q.WR:
        SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
FILL87:
        SRC/0.Q

COND.ADD&SHIFT:
        SRC/A.B,ALU/R.PLUS.S,DST/RSHF.RAM.Q,CIN/NO.CIN
COND.SUB&SHIFT:
        SRC/A.B,ALU/S.MINUS.R,DST/RSHF.RAM.Q,CIN/CIN
FILL8A:
        SRC/A.B
SHL2:
        SRC/A.B,ALU/R.PLUS.S,DST/LSHF.RAM,CIN/NO.CIN

ASHRC:
        SRC/0.B,ALU/R.OR.S,DST/RSHF.RAM.Q,CIN/NO.CIN
ASHR:
        SRC/0.B,ALU/R.OR.S,DST/RSHF.RAM,CIN/CIN
ASHLC:
        SRC/0.B,ALU/R.OR.S,DST/LSHF.RAM.Q,CIN/CIN
ASHL:
        SRC/0.B,ALU/R.OR.S,DST/LSHF.RAM,CIN/CIN

Q.PLUS.0:
        SRC/0.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
FILL91:
        SRC/0.Q
```

```
WR.CMP.Q:
        SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
Q.CMP.WR:
        SRC/A.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN

DEC.Q:
        SRC/0.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/NO.CIN
INC.Q:
        SRC/0.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/CIN
NEG.Q:
        SRC/0.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
COM.Q:
        SRC/0.Q,ALU/R.XNOR.S,DST/LOAD.Q,CIN/CIN

WR.BIT.WR:
        SRC/A.B,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
WR.CMP.WR:
        SRC/A.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
FILL9A:
        SRC/A.B
WR.PLUS.WR+1:
        SRC/A.B,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN

FILL9C:
        SRC/A.B
FILL9D:
        SRC/A.B
FILL9E:
        SRC/0.B
FILL9F:
        SRC/0.B

WR.PLUS.Q:
        SRC/A.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
WR.FROM.Q:
        SRC/A.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
Q.FROM.WR.Q:
        SRC/A.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
WR.OR.Q:
        SRC/A.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN

WR.AND.Q:
        SRC/A.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
WR.MASK.Q:
        SRC/A.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/CIN
WR.XOR.Q:
        SRC/A.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
FILLA7:
        SRC/A.Q

WR.PLUS.WR.Q:
        SRC/A.B,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
WR.FROM.WR.Q:
        SRC/A.B,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
FILLAA:
        SRC/A.B
WR.OR.WR.Q:
        SRC/A.B,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
```

```
WR.AND.WR.Q:
        SRC/A.B,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
WR.MASK.WR.Q:
        SRC/A.B,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/CIN
WR.XOR.WR.Q:
        SRC/A.B,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
FILLAF:
        SRC/A.B

Q.PLUS.WR:
        SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
WR.FROM.Q.WR:
        SRC/A.Q,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
Q.FROM.WR:
        SRC/A.Q,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
Q.OR.WR:
        SRC/A.Q,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN

Q.AND.WR:
        SRC/A.Q,ALU/R.AND.S,DST/WRITE.B.F,CIN/NO.CIN
FILLB5:
        SRC/A.Q
Q.XOR.WR:
        SRC/A.Q,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
Q.BIT.WR:
        SRC/A.Q,ALU/R.AND.S,DST/NOP,CIN/CIN

WR.PLUS.WR:
        SRC/A.B,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
WR.FROM.WR:
        SRC/A.B,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
WR.FROM.WR.WR:
        SRC/A.B,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
WR.OR.WR:
        SRC/A.B,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN

WR.FROM.WR-1:
        SRC/A.B,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
WR.MASK.WR:
        SRC/A.B,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
WR.XOR.WR:
        SRC/A.B,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
WR.AND.WR:
        SRC/A.B,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN

.PAGE
; THIS IS THE DP CODES FOR THE 'MOVE' MICRO INSTRUCTION
;
0C0:

MOV.MEM.WR:
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
```

```
SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
```

```
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN

MOV.LS.MEM:
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN

MOV.XWR.Q:
        SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN

MOV.LS.WR:
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN

.PAGE
MOV.MEM.LS:
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
MOV.ACC.LS:
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN


WR.PLUS.OS:
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN

MOV.WR.LS:
        SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
        SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN

.PAGE
; THIS GROUP OF INSTRUCTIONS IS FOR THE 'BASIC' MICRO INSTRUCTION
; THIS USES ADDRESSES 100-1FF

100:
LS.PLUS.WR:
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
LS.FROM.WR:
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
LS.AND.WR:
        SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
LS.XOR.WR:
        SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN

LS.FROM.WR-1:
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
LS.MASK.WR:
        SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
LS.PLUS.WR+1:
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
LS.OR.WR:
```

```
SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
```

```
        SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN

WR.PLUS.LS.Q:
        SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
LS.FROM.WR.Q:
        SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
WR.FROM.LS.Q:
        SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
WR.OR.LS.Q:
        SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN

WR.AND.LS.Q:
        SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
WR.XOR.LS.Q:
        SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
LS.CMP.WR:
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
WR.CMP.LS:
        SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN

LS.PLUS.Q:
        SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
LS.FROM.Q:
        SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
Q.FROM.LS.Q:
```

```
SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
```

```
        SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
LS.OR.Q:
        SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN

LS.MASK.Q:
        SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
LS.XOR.Q:
        SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
LS.AND.Q:
        SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
        SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
LS.BIT.Q:
        SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN

MOV.LS.Q:
        SRC/D.0,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
WR.BIT.LS:
        SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
LS.CMP.Q:
        SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
Q.CMP.LS:
        SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN

Q.PLUS.LS.TO.WR:
        SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
WRA.FROM.LS.WRB:
        SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
LS.NEG.WR:
        SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
LS.COM.WR:
        SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
        SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
.PAGE
;
; THE FOLLOWING LOCATIONS HAVE THE LOCAL STORE AS THEIR DESTINATION.
;
; THIS FACT IS USED BY HARDWARE TO GENERATE LOCAL STORE WRITE  PULSES

180:

LS.PLUS.WR.XCHG:
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
LS.FROM.WR.XCHG:
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
LS.AND.WR.XCHG:
        SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
LS.XOR.WR.XCHG:
        SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN

SWAP.LS.WR:
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
CLR.LS:
        SRC/D.0,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.AND.S,DST/NOP,CIN/CIN
MOV.Q.WR&WR.LS:
        SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
        SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
```

```
WR.PLUS.Q.XCHG:
        SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
        SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN

WR.FROM.LS-1:
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
LS.FROM.WR.LS:
        SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
WR.PLUS.LS+1:
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
WR.FROM.LS:
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN

WR.PLUS.LS:
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
WR.OR.LS:
        SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
WR.AND.LS:
        SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
WR.XOR.LS:
        SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
        SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN

Q.PLUS.LS:
        SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
LS.FROM.Q.LS:
        SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
```

```
Q.FROM.LS:
        SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
Q.OR.LS:
        SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN

Q.AND.LS:
        SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
        SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
        SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
        SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
Q.XOR.LS:
        SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
        SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
MOV.Q.LS:
        SRC/0.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/0.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/0.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
        SRC/0.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
Q.NEG.LS:
        SRC/0.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/0.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/0.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/0.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN

WR.COM.LS:
        SRC/0.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
        SRC/0.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
        SRC/0.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
        SRC/0.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
WR.NEG.LS:
        SRC/0.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/0.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/0.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/0.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
Q.PLUS.WR.TO.LS:
        SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
        SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
Q.FROM.WR.TO.LS:
        SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
        SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN

DEC.LS:
        SRC/D.0,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
        SRC/D.0,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
```

```
COM.LS:
        SRC/D.0,ALU/R.XNOR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.XNOR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.XNOR.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.XNOR.S,DST/NOP,CIN/CIN
NEG.LS:
        SRC/D.0,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.0,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.0,ALU/S.MINUS.R,DST/NOP,CIN/CIN
        SRC/D.0,ALU/S.MINUS.R,DST/NOP,CIN/CIN
INC.LS:
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
        SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN

.UCODE
.PAGE   "DIAGNOSTIC MACROS AND DEFINITIONS"

D.ADRS/=<15:9>,.VALIDITY=<D.VAL>,.DEFAULT=0

        SAV.WR0=1               ; STORAGE FOR WR0
        SAV.WR1=2               ; STORAGE FOR WR1
        SAV.WR2=3               ; STORAGE FOR WR2
        SAV.WR3=4               ; STORAGE FOR WR3
        T5.SUB=5                ; RESERVED FOR COMMON SUBROUTINES
        T6.SUB=6                ; RESERVED FOR COMMON SUBROUTINES
        T7.SUB=7                ; RESERVED FOR COMMON SUBROUTINES
        TRANSFER.POINTER=7      ; POINTER FOR TRANSFER ROUTINE
        MEMORY.SIZE=7           ; STORAGE FOR MEMORY SIZE IN 256KB BLOCKS AFTER
                                ;  SIZING
        FPA.FOUND=7             ; STORAGE FOR RESULT OF SIZING FOR FPA PRESENT
        UBE.FOUND=7             ; STORAGE FOR RESULT OF SIZING FOR UBE PRESENT
        T8=8                    ; TEMP LOCATION 8
        T9=9                    ; TEMP LOCATION 9
        T10=0A                  ; TEMP LOCATION 10
        T11=0B                  ; TEMP LOCATION 11
        T12=0C                  ; TEMP LOCATION 12
        T13=0D                  ; TEMP LOCATION 13
        T14=0E                  ; TEMP LOCATION 14
        T15=0F                  ; TEMP LOCATION 15
        PC=10                   ; MACRO PROGRAM COUNTER

        WR.BACKUP=11            ; BACKUP WR FOR MOV MEM.DATA TO WR[]
        MM.LASTADDR+1=12        ; STORAGE OF LAST ADDRESS IN MAIN MEMORY PLUS
                                ;  1 (FIRST NON-EXISTANT MEMORY ADDRESS)
        #FF=13                  ; MASK
        #FFFF=14                ; MASK
        #FF000000=15            ; MASK
        #FFFFFF00=16            ; MASK
        CSR0.MASK=16            ; MASK
        CSR1.MASK=17            ; MASK (01003FFF)
        CSR2.MASK=18            ; MASK (7FFE3FFF)
        #FE7FFFFF=19            ; MASK
        UBS.SET=1A              ; CONSTANT (7FF80000) USED BY UBS ADDRESS TEST
        UBS.DATA=1B             ; MASK (7FFF8000) USED AS ERROR MASK FOR UBS
                                ; DATA TEST
```

```
ERR.CON.NA=1E                ; CONSTANT OF 800500(H) BITS 23,10,8 SET FOR
                             ; LOADING CONTROL AND STATUS REG IN INITIAL
                             ; TESTS
ERR.CON=1F                   ; CONSTANT OF 500(H) BITS 10 & 8 SET FOR
                             ; LOADING CONTROL AND STATUS REG IN INITIAL
                             ; TESTS

#1=20                        ; PATTERN 00000001 (HEX)
#2=21                        ; PATTERN 00000002
#4=22                        ; PATTERN 00000004
#8=23                        ; PATTERN 00000008
#10=24                       ; PATTERN 00000010
#20=25                       ; PATTERN 00000020
#40=26                       ; PATTERN 00000040
#80=27                       ; PATTERN 00000080
#100=28                      ; PATTERN 00000100
#200=29                      ; PATTERN 00000200
#400=2A                      ; PATTERN 00000400
#800=2B                      ; PATTERN 00000800
#1000=2C                     ; PATTERN 00001000
#2000=2D                     ; PATTERN 00002000
#4000=2E                     ; PATTERN 00004000
#8000=2F                     ; PATTERN 00008000
#10000=30                    ; PATTERN 00010000
#20000=31                    ; PATTERN 00020000
#40000=32                    ; PATTERN 00040000
#80000=33                    ; PATTERN 00080000
#100000=34                   ; PATTERN 00100000
#200000=35                   ; PATTERN 00200000
#400000=36                   ; PATTERN 00400000
#800000=37                   ; PATTERN 00800000
#1000000=38                  ; PATTERN 01000000
#2000000=39                  ; PATTERN 02000000
#4000000=3A                  ; PATTERN 04000000
#8000000=3B                  ; PATTERN 08000000
#10000000=3C                 ; PATTERN 10000000
#20000000=3D                 ; PATTERN 20000000
#40000000=3E                 ; PATTERN 40000000
#80000000=3F                 ; PATTERN 80000000

BIT0=20                      ; PATTERN 00000001
BIT1=21                      ; PATTERN 00000002
BIT2=22                      ; PATTERN 00000004
BIT3=23                      ; PATTERN 00000008
BIT4=24                      ; PATTERN 00000010
BIT5=25                      ; PATTERN 00000020
BIT6=26                      ; PATTERN 00000040
BIT7=27                      ; PATTERN 00000080
BIT8=28                      ; PATTERN 00000100
BIT9=29                      ; PATTERN 00000200
BIT10=2A                     ; PATTERN 00000400
BIT11=2B                     ; PATTERN 00000800
BIT12=2C                     ; PATTERN 00001000
BIT13=2D                     ; PATTERN 00002000
BIT14=2E                     ; PATTERN 00004000
BIT15=2F                     ; PATTERN 00008000
BIT16=30                     ; PATTERN 00010000
```

```
        BIT17=31                        ; PATTERN 00020000
        BIT18=32                ; PATTERN 00040000
        BIT19=33                ; PATTERN 00080000
        BIT20=34                ; PATTERN 00100000
        BIT21=35                ; PATTERN 00200000
        BIT22=36                ; PATTERN 00400000
        BIT23=37                ; PATTERN 00800000
        BIT24=38                ; PATTERN 01000000
        BIT25=39                ; PATTERN 02000000
        BIT26=3A                ; PATTERN 04000000
        BIT27=3B                ; PATTERN 08000000
        BIT28=3C                ; PATTERN 10000000
        BIT29=3D                ; PATTERN 20000000
        BIT30=3E                ; PATTERN 40000000
        BIT31=3F                ; PATTERN 80000000


;CSR ADDRESS MNEMONICS

        CSR0=4E                 ; ALL BITS CLEAR TO ACCESS CSR 0
        CSR1=22                 ; BIT 2 SET TO ACCESS CSR 1
        CSR2=23                 ; BIT 3 SET TO ACCESS CSR 2

;CONTROL AND STATUS WORD BITS

        PRINT.UBE=26            ; PRINT IF UBE PRESENT OR NOT PRESENT (BIT 6)
                                ;  (INDICATOR IN LS 7)
        PRINT.R80=27            ; PRINT IF R80 PRESENT OR NOT PRESENT AND ON
                                ; WHICH DRIVE (BIT 7).  INDICATOR IN LS 7,
                                ; DRIVE NUMBER IN LS 6.
        ERROR=28                ; TEST ERROR INDICATION (BIT 8)
        SET.PA.ERR=29           ; TELLS CONSOLE TO CREATE A PARITY ERROR AT WCS
                                ;  ADDRESS POINTED TO BY LS 7 (BIT 9)
        CONSOLE.LOE=2A          ; INDICATES CONSOLE DOES ERROR LOOPING (BIT 10)
                                ;  (FOR INITIAL TESTS)
        PRINT.MEM.SIZE=2B       ; PRINT MEMORY SIZE IN 256K BLOCKS (BIT 11)
                                ;  (SIZE IN LS 7)
        PRINT.FPA=2C            ; PRINT IF FPA PRESENT OR NOT PRESENT (BIT 12)
                                ;  (INDICATOR IN LS 7)
        XFER.DATA=2D            ; INDICATES A DATA TRANSFER TO LS OR MM (BIT 13)
        EOS=2E                  ; END OF SECTION (BIT 14)
        BEGIN.TEST=2F           ; BEGINNING OF TEST (BIT 15)
        INTERUPT.EN=30          ; ENABLE INTERUPT OR SIGNAL SPECIFIED BY BITS
                                ;  17 THRU 22 AND 28 THRU 30 (BIT 16)

         CONSOLE.HALT=31        ;   CONSOLE HALT INTERRUPT (BIT 17)
         PWR.FAIL=32            ;   PWR FAIL INT (BIT 18)
         INTERVAL.TIM=33        ;   INTRVL TIM INT (BIT 19)
         CONSOLE.ATTN=34        ;   CONSOLE ATTN INTERRUPT (BIT 20)
         CONSOLE.ACK=35         ;   CONSOLE ACK INTERRUPT (BIT 21)
         DISABLE.MEM.REF=36     ;   DISABLE MEMORY REFERANCES (BIT 22)
         CINIT=3C               ;   UNIBUS INIT SIGNAL TO THE MCT (BIT 28)
         UBS.DCLO=3D            ;   UNIBUS DC LO INDICATION TO THE MCT (BIT 29)
         UBS.BBSY=3E            ;   UNIBUS BUS BUSY INDICATOR TO MCT (BIT 30)

        NA.EXP.REC=37           ; PRINT N/A UNDER EXPECTED AND RECEIVED (BIT 23)
        OTHER.DATA=38           ; PRINT A VALUE UNDER OTHER DATA (BIT 24)
        CONSOLE.LOOP=39         ; CONSOLE PERFORMS A LOOP ACCORDING TO LOOP
```

```
                    ; COUNT IN LS (BIT 25)
```

```
        PRINT.CRD=3A            ; PRINT NUMBER OF SINGLE BIT ERRORS (BIT 26)
        CLR.PA.ERR=3B           ; TELLS CONSOLE TO CLEAR PARITY ERROR AT WCS
                                ;  ADDRESS POINTED TO BY LS 7 (BIT 27)
        PRINT.IDC=3F            ; PRINT IF IDC PRESENT OR NOT PRESENT (BIT 31)
                                ;  (INDICATOR IN LS 7)

;MODULE CODES

        WCS=20                  ; MODULE CODE FOR WCS BOARD (BIT 0 SET)
        CPU=21                  ; MODULE CODE FOR CPU BOARD (BIT 1 SET)
        MCT=22                  ; MODULE CODE FOR MCT BOARD (BIT 2 SET)
        FPA=23                  ; MODULE CODE FOR FPA BOARD (BIT 3 SET)
        ARRAY=24                ; MODULE CODE FOR ARRAY BOARD (BIT 4 SET)
        IDC=25                  ; MODULE CODE FOR IDC BOARD (BIT 5 SET)

;CSR 1 ERROR BITS

        VALID.ERR=2E            ; VALID NOT SET IF 1 (BIT 14)
        TB.PAR.ERR=2F           ; TB PARITY ERROR (BIT 15)
        NXM=30                  ; NON-EXISTANT MEMORY ERROR (BIT 16)
        UB.BSY=31               ; UNIBUS BUSY (BIT 17)
        ADP.REG=32              ; UNIBUS ADAPTER REGISTER SELECT (BIT 18)
        WR.ACROSS.PG=33         ; WRITE ACROSS PAGE BOUNDARY ERROR (BIT 19)
        OP.ERR=34               ; OPERATION ERROR (BIT 20)
        TB.MISS=35              ; TB MISS (VIRTUAL ADDRESS TRANSLATION NOT IN
                                ;   BUFFER) (BIT 21)
        ACCESS.REFUSED=36       ; PROTECTION ERROR ON ACCESS (BIT 22)
        MODIFY.REFUSED=37       ; PROTECTION ERROR ON WRITE (BIT 23)

        CRD=3E                  ; SINGLE BIT ERROR CORRECTED (BIT 30)
        RDS=3F                  ; UNCORRECTABLE DATA ERROR (BIT 31)


;CSR 1 CONTROL BITS

        ECC.DIS=39              ; CSR1 ECC DISABLE BIT (BIT 25)
        DIAG.CHK=3A             ; CSR1 DIAG CHK BIT (BIT 26)
        MME=3B                  ; CSR1 MEMORY MANAGMENT ENABLE BIT (BIT 27)
        INH.CRD=3C              ; CSR1 INHIBIT REPORT CRD BIT (BIT 28)
        TB.PAR.DIAG=3D          ; CSR1 TB PAR DIAG BIT (FORCE TB PARITY ERR)
                                ;  (BIT 29)


;UBE CONTROL REGISTER BITS

  ;CONTROL REG 1
        GO=20                   ; START UBE (BIT 0)
        BR4=21                  ; ISSUE BUS REQUEST 4 (BIT 1)
        BR5=22                  ; ISSUE BUS REQUEST 5 (BIT 2)
        BR6=23                  ; ISSUE BUS REQUEST 6 (BIT 3)
        BR7=24                  ; ISSUE BUS REQUEST 7 (BIT 4)
        NPR=25                  ; ISSUE NPR (BIT 5)
        INT.ODNE=26             ; INTERRUPT ON DONE (WHEN CCOVF) (BIT 6)
        RDY=27                  ; READY BIT, SET WHEN READY TO BEGIN FUNC (BIT 7)
        C00=28                  ; C0 BIT (BIT 8)
        C01=29                  ; C1 BIT (BIT 9)
        FUNA=2A                 ; FUNCTION BIT A (BIT 10
        FUNB=2B                         ; FUNCTION BIT B (BIT 11)
        CC+DAT=2C               ; DATA FROM CYCLE COUNT IF SET, DATA REG IF CLEAR
        INH.DATIP.ROL=2D        ; INHIBIT ROL ON DATIP (BIT 13)
        DATOB.ON.DATIP=2E       ; DO DATOB AFTER DATIP CYCLE (BIT 14)
        ERR=2F                  ; EXERCISOR OR UNIBUS ERROR (BIT 15)

  ;CONTROL REG 2
        EXT.MEM0=20             ; EXTENDED MEMORY BIT 0 (BIT 0)
        EXT.MEM1=21             ; EXTENDED MEMORY BIT 1 (BIT 1)
        INH.INC.ADDR&CC=22      ; INHIBIT INC ON CYCLE COUNT AND ADDR REG (BIT 2)
        INH.SACK=23             ; INHIBIT ASSERING BUS SACK ON BUS GRANT (BIT 3)
        PWR.DWN.SEQ=24          ; ASSERT ACLO (BIT 4)
        WNG.GNT.BCK=25          ; NO BUS GRANT RECV TO HIGEST REQUEST (BIT 5)
        MAX.LATE.ERR=26         ; NPR AND BR LATENCY TIME EXCEEDED (BIT 6)
        NO.SACK.ERR=27          ; NO TIMEOUT AT CPU WHEN NO SACK (BIT 7)
        NO.SSYN.ERR=28          ; NO SSYN RECEIVED AFTER ASSERTING MSYN (BIT 8)
        WRG.A.LINES=29          ; ADDRESS ERR ON SENT & RECEIVED ADDRESS (BIT 9)
        NO.GNT+NOT.ONE.GNT=2A   ; NO GRANT OR MORE THAN 1 GRANT AFTER REQ. (BIT 10)
        INTR.SSYN.ERR=2B        ; NO SSYN RECEIVED AFTER BUS INTR (BIT 11)
        ENB.PB=2C               ; ENABLE PARITY BIT B (BIT 12)
        CCOVF=2D                ; CYCLE COUNT OVERFLOW (BIT 13)
        TM.DLY=2E               ; REQUEST BUS EVERY 10 USEC (BIT 14)

;BG6 MNEMONIC
        BG6=23                  ; BIT 3 SET FOR BG 6 ADDRESS

;ERROR AND CONTROL INFORMATION

        CONTROL.STATUS=40       ; CONTROL AND STATUS WORD
        ERROR.NUMBER=41         ; ERROR NUMBER OF CURRENT TEST
        EXPECTED.DATA=42        ; EXPECTED RESULT OF CURRENT TEST
        RECEIVED.DATA=43        ; ACTUAL RESULT OF CURRENT TEST
        ADDRESS.DATA=44         ; OTHER PERTINENT DATA
        ERROR.MASK=45           ; BITS TO CHECK IN RESULT
        MODULE.NUM=46           ; STORAGE OF MODULE NUMBER (CODED)
        LOOP.CNT=47             ; STORAGE OF LOOP COUNT FOR CONSOLE LOOP
                                ; CONTROL.
        ERROR.CONTROL=48        ; STORAGE FOR ERROR CONTROL (LOOP ON ERROR ETC.)
         LOE=20                 ;  LOOP ON ERROR IF SET (BIT0)
         NER=21                 ;  NO ERROR REPORT IF SET (BIT1)
         BELL=22                ;  BELL ON ERROR IF SET (BIT2)
         HOE=23                 ;  HALT ON ERROR IF SET (BIT3)
         SER=24                 ;  ENABLE ERROR REPORT ON CRD ERRORS IF SET
         APT.PRESENT=25         ;  APT PRESENT IF SET (BIT5)
         LOOP.COMMAND=26        ;  LOOP COMMAND TYPED IF SET (BIT 6)

        APT.PARAM=49            ;APT RUN TIME PARAMETER REGS (MEM SIZE, HARD-
                                ; WARE CONFIG, SOFTWARE CONFIG IN BYTES 0, 1
                                ; AND 2 RESPECTIVELY. BYTE 3 FOR FUTURE USE)
        APT.RESERVED=4A         ;RESERVED FOR FUTURE APT USE

;MISCELLANEOUS CONSTANTS AND STORAGE

        #AAAAAAAA=4C            ; CONSTANT
        ALTER.ODD=4C            ; CONSTANT
        #55555555=4D            ; CONSTANT
        ALTER.EVEN=4D           ; CONSTANT
        #0=4E                   ; CONSTANT
        ZERO=4E                 ; CONSTANT
        #FFFFFFFF=4F                ; CONSTANT
        ONES=4F                     ; CONSTANT
        PREVIOUS.ERROR=50           ; ERROR NUMBER OF LAST ERROR

        DATA.1=51                   ; STORAGE FOR FPA DATA
        DATA.2=52                   ; STORAGE FOR FPA DATA
        DATA.3=53                   ; STORAGE FOR FPA DATA
        FIRST.FLT=54                ; STORAGE FOR FPA DATA
        SEC.FLT=55                  ; STORAGE FOR FPA DATA
        THIRD.FLT=56                ; STORAGE FOR FPA DATA
        T57=57                      ; TEMP LOCATION 57
        T58=58                      ; TEMP LOCATION 58
        T59=59                      ; TEMP LOCATION 59

        BEDB=5A                     ;UBE (BUS EXERCISOR) DATA BUF REG
        BECC=5B                     ;UBE (BUS EXERCISOR) CYCLE COUNT REG
        BEBA=5C                     ;UBE (BUS EXERCISOR) ADDR REG
        BECR1=5D                    ;UBE (BUS EXERCISOR) CONTROL REG 1
        CLEAR.ERR.ADDR=5E           ;UBE CLEAR ERROR REG ADDRESS
        BECR2=5F                    ;UBE (BUS EXERCISOR) CONTROL REG 2

        #3(H)=60                    ; CONSTANT
        #12(H)=61                   ; CONSTANT
        #33333333=62                ; CONSTANT
        #0F0F0F0F=63                ; CONSTANT
        #00FF00FF=64                ; CONSTANT
        #162(H)=65                  ; CONSTANT FOR LS TRANSFER POINTER
        BR7.IDENT=66                ; BR7 IDENTIFIER (1010(B) IN BITS 5-2)
        BG7=67                      ; BG7 ADDRESS (BITS 2 AND 3 SET)
;TEMPS FOR CPU TESTS
        L30=30                      ;LS 30 TEMP (RESTORED TO 10000 AFTER USE)
        L31=31                      ;LS 31 TEMP (RESTORE TO 20000 AFTER USE)
        L53=53                      ;LS 53 TEMP
        L73=73                      ;LS 73 TEMP

;REGISTER ADDRESSES

        CONTROL.OS=78               ; HARDWARE INDEX TO CONTROL WORDS (LS 40-4F)
        SHIFT.OS(3-0)=79            ; 16 LOCATION HARDWARE INDEX TO SHIFT PATTERN
                                    ;  (LS 20-2F)
        SHIFT.OS(4-0)=7B            ; 32 LOCATION HARDWARE INDEX TO SHIFT PATTERN
                                    ;  (LS 20-3F)
        OS=7C                       ; OS REGISTER
        DEC.CON=7D                  ; DECIMAL CARRIES
        SIZE=7E                     ; SIZE REGISTER
        MDT= 7E                     ; MEMORY REFERENCE DATA SIZE
        INTERRUPT.VEC=7E            ; HARDWARE INTERRUPT VECTOR
;
;       THIS WORD IS THE HARDWARE CONDITION CODES.  THE CC'S CAN BE READ
;       OR ARE WRITTEN HERE.  NO OTHER BITS IN THE PSL ARE EFFECTED.
;
        PSL.CC=7F                   ; PSL CONDITION CODES
.PAGE
XD.ADRS/=<16:9>,.VALIDITY=<XD.VAL>
```

```
        SAV.WR0=1                       ; STORAGE FOR WR0
```

```
SAV.WR1=2                ; STORAGE FOR WR1
SAV.WR2=3                ; STORAGE FOR WR2
SAV.WR3=4                ; STORAGE FOR WR3
T5.SUB=5                 ; RESERVED FOR COMMON SUBROUTINES
T6.SUB=6                 ; RESERVED FOR COMMON SUBROUTINES
T7.SUB=7                 ; RESERVED FOR COMMON SUBROUTINES
TRANSFER.POINTER=7       ; POINTER FOR TRANSFER ROUTINE
MEMORY.SIZE=7            ; STORAGE FOR MEMORY SIZE IN 256KB BLOCKS AFTER
                         ;  SIZING
FPA.FOUND=7              ; STORAGE FOR RESULT OF SIZING FOR FPA PRESENT
UBE.FOUND=7              ; STORAGE FOR RESULT OF SIZING FOR UBE PRESENT
T8=8                     ; TEMP LOCATION 8
T9=9                     ; TEMP LOCATION 9
T10=0A                   ; TEMP LOCATION 10
T11=0B                   ; TEMP LOCATION 11
T12=0C                   ; TEMP LOCATION 12
T13=0D                   ; TEMP LOCATION 13
T14=0E                   ; TEMP LOCATION 14
T15=0F                   ; TEMP LOCATION 15
PC=10                    ; MACRO PROGRAM COUNTER

WR.BACKUP=11             ; BACKUP WR FOR MOV MEM.DATA TO WR[]
MM.LASTADDR+1=12         ; STORAGE OF LAST ADDRESS IN MAIN MEMORY PLUS
                         ;  1 (FIRST NON-EXISTANT MEMORY ADDRESS)
#FF=13                   ; MASK
#FFFF=14                 ; MASK
#FF000000=15             ; MASK
#FFFFFF00=16             ; MASK
CSR0.MASK=16             ; MASK
CSR1.MASK=17             ; MASK (01003FFF)
CSR2.MASK=18             ; MASK (FFFE3FFF)
#FE7FFFFF=19             ; MASK
UBS.SET=1A               ; CONSTANT (7FF80000) USED BY UBS ADDRESS TEST
UBS.DATA=1B              ; MASK (7FFF8000) USED AS ERROR MASK FOR UBS
                         ; DATA TEST

ERR.CON.NA=1E            ; CONSTANT OF 800500(H) BITS 23,10,8 SET FOR
                         ; LOADING CONTROL AND STATUS REG IN INITIAL
                         ; TESTS
ERR.CON=1F               ; CONSTANT OF 500(H) BITS 10 & 8 SET FOR
                         ; LOADING CONTROL AND STATUS REG IN INITIAL
                         ; TESTS

#1=20                    ; PATTERN 00000001 (HEX)
#2=21                    ; PATTERN 00000002
#4=22                    ; PATTERN 00000004
#8=23                    ; PATTERN 00000008
#10=24                   ; PATTERN 00000010
#20=25                   ; PATTERN 00000020
#40=26                   ; PATTERN 00000040
#80=27                   ; PATTERN 00000080
#100=28                  ; PATTERN 00000100
#200=29                  ; PATTERN 00000200
#400=2A                  ; PATTERN 00000400
#800=2B                  ; PATTERN 00000800
#1000=2C                 ; PATTERN 00001000
#2000=2D                 ; PATTERN 00002000
```

```
        #4000=2E                    ; PATTERN 00004000
        ^X8000=2F               ; PATTERN 00008000
        ^X10000=30              ; PATTERN 00010000
        ^X20000=31              ; PATTERN 00020000
        ^X40000=32              ; PATTERN 00040000
        ^X80000=33              ; PATTERN 00080000
        ^X100000=34             ; PATTERN 00100000
        ^X200000=35             ; PATTERN 00200000
        ^X400000=36             ; PATTERN 00400000
        ^X800000=37             ; PATTERN 00800000
        ^X1000000=38            ; PATTERN 01000000
        ^X2000000=39            ; PATTERN 02000000
        ^X4000000=3A            ; PATTERN 04000000
        ^X8000000=3B            ; PATTERN 08000000
        ^X10000000=3C           ; PATTERN 10000000
        ^X20000000=3D           ; PATTERN 20000000
        ^X40000000=3E           ; PATTERN 40000000
        ^X80000000=3F           ; PATTERN 80000000

        BIT0=20                 ; PATTERN 00000001
        BIT1=21                 ; PATTERN 00000002
        BIT2=22                 ; PATTERN 00000004
        BIT3=23                 ; PATTERN 00000008
        BIT4=24                 ; PATTERN 00000010
        BIT5=25                 ; PATTERN 00000020
        BIT6=26                 ; PATTERN 00000040
        BIT7=27                 ; PATTERN 00000080
        BIT8=28                 ; PATTERN 00000100
        BIT9=29                 ; PATTERN 00000200
        BIT10=2A                ; PATTERN 00000400
        BIT11=2B                ; PATTERN 00000800
        BIT12=2C                ; PATTERN 00001000
        BIT13=2D                ; PATTERN 00002000
        BIT14=2E                ; PATTERN 00004000
        BIT15=2F                ; PATTERN 00008000
        BIT16=30                ; PATTERN 00010000
        BIT17=31                ; PATTERN 00020000
        BIT18=32                ; PATTERN 00040000
        BIT19=33                ; PATTERN 00080000
        BIT20=34                ; PATTERN 00100000
        BIT21=35                ; PATTERN 00200000
        BIT22=36                ; PATTERN 00400000
        BIT23=37                ; PATTERN 00800000
        BIT24=38                ; PATTERN 01000000
        BIT25=39                ; PATTERN 02000000
        BIT26=3A                ; PATTERN 04000000
        BIT27=3B                ; PATTERN 08000000
        BIT28=3C                ; PATTERN 10000000
        BIT29=3D                ; PATTERN 20000000
        BIT30=3E                ; PATTERN 40000000
        BIT31=3F                ; PATTERN 80000000

;CSR ADDRESS MNEMONICS

        CSR0=4E                 ; ALL BITS CLEAR TO ACCESS CSR 0
        CSR1=22                 ; BIT 2 SET TO ACCESS CSR 1
        CSR2=23                 ; BIT 3 SET TO ACCESS CSR 2
```

```
;CONTROL AND STATUS WORD BITS

        PRINT.UBE=26            ; PRINT IF UBE PRESENT OR NOT PRESENT (BIT 6)
                                ;  (INDICATOR IN LS 7)
        PRINT.R80=27            ; PRINT IF R80 PRESENT OR NOT PRESENT AND ON
                                ; WHICH DRIVE (BIT 7).  INDICATOR IN LS 7,
                                ; DRIVE NUMBER IN LS 6.
        ERROR=28                ; TEST ERROR INDICATION (BIT 8)
        SET.PA.ERR=29           ; TELLS CONSOLE TO CREATE A PARITY ERROR AT WCS
                                ;  ADDRESS POINTED TO BY LS 7 (BIT 9)
        CONSOLE.LOE=2A          ; INDICATES CONSOLE DOES ERROR LOOPING (BIT 10)
                                ;  (FOR INITIAL TESTS)
        PRINT.MEM.SIZE=2B       ; PRINT MEMORY SIZE IN 256K BLOCKS (BIT 11)
                                ;  (SIZE IN LS 7)
        PRINT.FPA=2C            ; PRINT IF FPA PRESENT OR NOT PRESENT (BIT 12)
                                ;  (INDICATOR IN LS 7)
        XFER.DATA=2D            ; INDICATES A DATA TRANSFER TO LS OR MM (BIT 13)
        EOS=2E                  ; END OF SECTION (BIT 14)
        BEGIN.TEST=2F           ; BEGINNING OF TEST (BIT 15)
        INTERUPT.EN=30          ; ENABLE INTERUPT OR SIGNAL SPECIFIED BY BITS
                                ;  17 THRU 22 AND 28 THRU 30 (BIT 16)

         CONSOLE.HALT=31        ;   CONSOLE HALT INTERRUPT (BIT 17)
         PWR.FAIL=32            ;   PWR FAIL INT (BIT 18)
         INTERVAL.TIM=33        ;   INTRVL TIM INT (BIT 19)
         CONSOLE.ATTN=34        ;   CONSOLE ATTN INTERRUPT (BIT 20)
         CONSOLE.ACK=35         ;   CONSOLE ACK INTERRUPT (BIT 21
         DISABLE.MEM.REF=36     ;   DISABLE MEMORY REFERANCES (BIT 22)
         CINIT=3C               ;   UNIBUS INIT SIGNAL TO THE MCT (BIT 28)
         UBS.DCLO=3D            ;   UNIBUS DC LO INDICATION TO THE MCT (BIT 29)
         UBS.BBSY=3E            ;   UNIBUS BUS BUSY INDICATOR TO MCT (BIT 30)

        NA.EXP.REC=37           ; PRINT N/A UNDER EXPECTED AND RECEIVED (BIT 23)
        OTHER.DATA=38           ; PRINT A VALUE UNDER OTHER DATA (BIT 24)
        CONSOLE.LOOP=39         ; CONSOLE PERFORMS A LOOP ACCORDING TO LOOP
                                ; COUNT IN LS (BIT 25)
        PRINT.CRD=3A            ; PRINT NUMBER OF SINGLE BIT ERRORS (BIT 26)
        CLR.PA.ERR=3B           ; TELLS CONSOLE TO CLEAR PARITY ERROR AT WCS
                                ;  ADDRESS POINTED TO BY LS 7 (BIT 27)
        PRINT.IDC=3F            ; PRINT IF IDC PRESENT OR NOT PRESENT (BIT 31)
                                ;  (INDICATOR IN LS 7)

;MODULE CODES

        WCS=20                  ; MODULE CODE FOR WCS BOARD (BIT 0 SET)
        CPU=21                  ; MODULE CODE FOR CPU BOARD (BIT 1 SET)
        MCT=22                  ; MODULE CODE FOR MCT BOARD (BIT 2 SET)
        FPA=23                  ; MODULE CODE FOR FPA BOARD (BIT 3 SET)
        ARRAY=24                ; MODULE CODE FOR ARRAY BOARD (BIT 4 SET)
        IDC=25                  ; MODULE CODE FOR IDC BOARD (BIT 5 SET)

;CSR 1 ERROR BITS

        VALID.ERR=2E            ; VALID NOT SET IF 1 (BIT 14)
        TB.PAR.ERR=2F           ; TB PARITY ERROR (BIT 15)
```

```
	NXM=30			; NON-EXISTANT MEMORY ERROR (BIT 16)
```

```
        UB.BSY=31                   ; UNIBUS BUSY (BIT 17)
        ADP.REG=32                  ; UNIBUS ADAPTER REGISTER SELECT (BIT 18)
        WR.ACROSS.PG=33             ; WRITE ACROSS PAGE BOUNDARY ERROR (BIT 19)
        OP.ERR=34                   ; OPERATION ERROR (BIT 20)
        TB.MISS=35                  ; TB MISS (VIRTUAL ADDRESS TRANSLATION NOT IN
                                    ;   BUFFER) (BIT 21)
        ACCESS.REFUSED=36           ; PROTECTION ERROR ON ACCESS (BIT 22)
        MODIFY.REFUSED=37           ; PROTECTION ERROR ON WRITE (BIT 23)

        CRD=3E                      ; SINGLE BIT ERROR CORRECTED (BIT 30)
        RDS=3F                      ; UNCORRECTABLE DATA ERROR (BIT 31)


;CSR 1 CONTROL BITS

        ECC.DIS=39                  ; CSR1 ECC DISABLE BIT (BIT 25)
        DIAG.CHK=3A                 ; CSR1 DIAG CHK BIT (BIT 26)
        MME=3B                      ; CSR1 MEMORY MANAGMENT ENABLE BIT (BIT 27)
        INH.CRD=3C                  ; CSR1 INHIBIT REPORT CRD BIT (BIT 28)
        TB.PAR.DIAG=3D              ; CSR1 TB PAR DIAG BIT (FORCE TB PARITY ERR)
                                    ;  (BIT 29)


;UBE CONTROL REGISTER BITS

    ;CONTROL REG 1
        GO=20                       ; START UBE (BIT 0)
        BR4=21                      ; ISSUE BUS REQUEST 4 (BIT 1)
        BR5=22                      ; ISSUE BUS REQUEST 5 (BIT 2)
        BR6=23                      ; ISSUE BUS REQUEST 6 (BIT 3)
        BR7=24                      ; ISSUE BUS REQUEST 7 (BIT 4)
        NPR=25                      ; ISSUE NPR (BIT 5)
        INT.ODNE=26                 ; INTERRUPT ON DONE (WHEN CCOVF) (BIT 6)
        RDY=27                      ; READY BIT, SET WHEN READY TO BEGIN FUNC (BIT 7)
        C00=28                      ; C0 BIT (BIT 8)
        C01=29                      ; C1 BIT (BIT 9)
        FUNA=2A                     ; FUNCTION BIT A (BIT 10
        FUNB=2B                     ; FUNCTION BIT B (BIT 11)
        CC+DAT=2C                   ; DATA FROM CYCLE COUNT IF SET, DATA REG IF CLEAR
        INH.DATIP.ROL=2D            ; INHIBIT ROL ON DATIP (BIT 13)
        DATOB.ON.DATIP=2E           ; DO DATOB AFTER DATIP CYCLE (BIT 14)
        ERR=2F                      ; EXERCISOR OR UNIBUS ERROR (BIT 15)

    ;CONTROL REG 2
        EXT.MEM0=20                 ; EXTENDED MEMORY BIT 0 (BIT 0)
        EXT.MEM1=21                 ; EXTENDED MEMORY BIT 1 (BIT 1)
        INH.INC.ADDR&CC=22          ; INHIBIT INC ON CYCLE COUNT AND ADDR REG (BIT 2)
        INH.SACK=23                 ; INHIBIT ASSERING BUS SACK ON BUS GRANT (BIT 3)
        PWR.DWN.SEQ=24              ; ASSERT ACLO (BIT 4)
        WNG.GNT.BCK=25              ; NO BUS GRANT RECV TO HIGEST REQUEST (BIT 5)
        MAX.LATE.ERR=26             ; NPR AND BR LATENCY TIME EXCEEDED (BIT 6)
        NO.SACK.ERR=27              ; NO TIMEOUT AT CPU WHEN NO SACK (BIT 7)
        NO.SSYN.ERR=28              ; NO SSYN RECEIVED AFTER ASSERTING MSYN (BIT 8)
        WRG.A.LINES=29              ; ADDRESS ERR ON SENT & RECEIVED ADDRESS (BIT 9)
        NO.GNT+NOT.ONE.GNT=2A       ; NO GRANT OR MORE THAN 1 GRANT AFTER REQ. (BIT 10)
        INTR.SSYN.ERR=2B            ; NO SSYN RECEIVED AFTER BUS INTR (BIT 11)
        ENB.PB=2C                   ; ENABLE PARITY BIT B (BIT 12)
        CCOVF=2D                  ; CYCLE COUNT OVERFLOW (BIT 13)
        TM.DLY=2E               ; REQUEST BUS EVERY 10 USEC (BIT 14)

;BG6 MNEMONIC
        BG6=23                  ; BIT 3 SET FOR BG 6 ADDRESS

;ERROR AND CONTROL INFORMATION

        CONTROL.STATUS=40       ; CONTROL AND STATUS WORD
        ERROR.NUMBER=41         ; ERROR NUMBER OF CURRENT TEST
        EXPECTED.DATA=42        ; EXPECTED RESULT OF CURRENT TEST
        RECEIVED.DATA=43        ; ACTUAL RESULT OF CURRENT TEST
        ADDRESS.DATA=44         ; OTHER PERTINENT DATA
        ERROR.MASK=45           ; BITS TO CHECK IN RESULT
        MODULE.NUM=46           ; STORAGE OF MODULE NUMBER (CODED)
        LOOP.CNT=47             ; STORAGE OF LOOP COUNT FOR CONSOLE LOOP
                                ; CONTROL.
        ERROR.CONTROL=48        ; STORAGE FOR ERROR CONTROL (LOOP ON ERROR ETC.)
         LOE=20                 ;  LOOP ON ERROR IF SET (BIT0)
         NER=21                 ;  NO ERROR REPORT IF SET (BIT1)
         BELL=22                ;  BELL ON ERROR IF SET (BIT2)
         HOE=23                 ;  HALT ON ERROR IF SET (BIT3)
         SER=24                 ;  ENABLE ERROR REPORT ON CRD ERRORS IF SET
         APT.PRESENT=25         ;  APT PRESENT IF SET (BIT5)
         LOOP.COMMAND=26        ;  LOOP COMMAND TYPED IF SET (BIT 6)

        APT.PARAM=49            ;APT RUN TIME PARAMETER REGS (MEM SIZE, HARD-
                                ; WARE CONFIG, SOFTWARE CONFIG IN BYTES 0, 1
                                ; AND 2 RESPECTIVELY. BYTE 3 FOR FUTURE USE)
        APT.RESERVED=4A         ;RESERVED FOR FUTURE APT USE

;MISCELLANEOUS CONSTANTS AND STORAGE

        #AAAAAAAA=4C            ; CONSTANT
        ALTER.ODD=4C            ; CONSTANT
        #55555555=4D            ; CONSTANT
        ALTER.EVEN=4D           ; CONSTANT
        #0=4E                   ; CONSTANT
        ZERO=4E                 ; CONSTANT
        #FFFFFFFF=4F            ; CONSTANT
        ONES=4F                 ; CONSTANT
        PREVIOUS.ERROR=50       ; ERROR NUMBER OF LAST ERROR

        DATA.1=51               ; STORAGE FOR FPA DATA
        DATA.2=52               ; STORAGE FOR FPA DATA
        DATA.3=53               ; STORAGE FOR FPA DATA
        FIRST.FLT=54            ; STORAGE FOR FPA DATA
        SEC.FLT=55              ; STORAGE FOR FPA DATA
        THIRD.FLT=56            ; STORAGE FOR FPA DATA
        T57=57                  ; TEMP LOCATION 57
        T58=58                  ; TEMP LOCATION 58
        T59=59                  ; TEMP LOCATION 59

        BEDB=5A                 ;UBE (BUS EXERCISOR) DATA BUF REG
        BECC=5B                 ;UBE (BUS EXERCISOR) CYCLE COUNT REG
        BEBA=5C                 ;UBE (BUS EXERCISOR) ADDR REG
        BECR1=5D                ;UBE (BUS EXERCISOR) CONTROL REG 1
        CLEAR.ERR.ADDR=5E       ;UBE CLEAR ERROR REG ADDRESS
```

```
	BECR2=5F			;UBE (BUS EXERCISOR) CONTROL REG 2
```

```
        #3(H)=60                        ; CONSTANT
        #12(H)=61                       ; CONSTANT
        #33333333=62                    ; CONSTANT
        #0F0F0F0F=63                    ; CONSTANT
        #00FF00FF=64                    ; CONSTANT
        #162(H)=65                      ; CONSTANT FOR LS TRANSFER POINTER
        BR7.IDENT=66                    ; BR7 IDENTIFIER (1010(B) IN BITS 5-2)
        BG7=67                          ; BG7 ADDRESS (BITS 2 AND 3 SET)

;TEMPS FOR CPU TESTS
        L30=30                          ;LS 30 TEMP (RESTORED TO 10000 AFTER USE)
        L31=31                          ;LS 31 TEMP (RESTORE TO 20000 AFTER USE)
        L53=53                          ;LS 53 TEMP
        L73=73                          ;LS 73 TEMP

;REGISTER ADDRESSES

        CONTROL.OS=78                   ; HARDWARE INDEX TO CONTROL WORDS (LS 40-4F)
        SHIFT.OS(3-0)=79                ; 16 LOCATION HARDWARE INDEX TO SHIFT PATTERN
                                        ;  (LS 20-2F)
        SHIFT.OS(4-0)=7B                ; 32 LOCATION HARDWARE INDEX TO SHIFT PATTERN
                                        ;  (LS 20-3F)
        OS=7C                           ; OS REGISTER
        DEC.CON=7D                      ; DECIMAL CARRIES
        SIZE=7E                         ; SIZE REGISTER
        MDT= 7E                         ; MEMORY REFERENCE DATA SIZE
        INTERRUPT.VEC=7E                ; HARDWARE INTERRUPT VECTOR
;
;       THIS WORD IS THE HARDWARE CONDITION CODES.  THE CC'S CAN BE READ
;       OR ARE WRITTEN HERE.  NO OTHER BITS IN THE PSL ARE EFFECTED.
;
        PSL.CC=7F                       ; PSL CONDITION CODES


.TOC    "COMMON LS ASSIGNMENTS (TO REGISTERS)"
;
;UPPER HALF OF LS
;
        XGPR.OS=0F8                     ; HARDWARE INDEX TO XGPRS
        USER.INDEX(3-0)=0F9             ; 16 LOCATION HARDWARE INDEX TO DIAGNOSTIC DATA
                                        ;    AREA (LS LOCATIONS 0A0 THROUGH 0AF)
        USER.INDEX(4-0)=0FB             ; 32 LOCATION HARDWARE INDEX TO DIAGNOSTIC DATA
                                        ;    AREA (LS LOCATIONS 0A0 THROUGH 0BF)
        CCR=0FC                         ; CONSOLE READ REGISTER
        TIMER=0FD                       ; INTERVAL TIMER VALUE.
        ALU.CC=0FE                      ; ALU CONDITION CODES
;
;       THIS LOCATION IS A WRITE ONLY REGISTER.  THE FOLLOWING BITS IN THE
;       PSL ARE AFFECTED:
;
;       COMPATIBILITY MODE - BIT<31>
;       CURRENT MODE - BITS<25:24>
;       INTERRUPT PRIORITY LEVEL (IPL) - BITS<20:16>
;       TRACE TRAP ENABLE (REALLY T OR TP) - BIT<4>
;       NEGATIVE CONDITION CODE - BIT<3>
;       ZERO CONDITION CODE - BIT<2>
;       OVERFLOW CONDITION CODE - BIT<1>
;       CARRY CONDITION CODE - BIT<0>
;
        PSL.HW=0FF               ; HARDWARE PSL BITS
.TOC    "PROGRAM SPECIFIC LS ASSIGNMENTS (LS 80-F7)"
;
; THESE ADDRESSES ARE ONLY ACCESSIBLE WITH THE MOV MICRO INSTRUCTION.
; STARTING AT ADDRESS 80(H).  THEY ARE NOT USED BY ALL SOFTWARE MODULES
; BUT ARE SPECIFIC TO THIS ONE.
;


        L90=90                          ;TEMP LOCATION
        ALTER.MIX=91                    ;DATA: AAAA5555
        HI.IPL=92                       ;BITS 16-20 SET (IPL=1F)
        #FFFFFFFC=95                    ;FREQUENTLY USED MASK (BOTTOM BYTE)
        #000F=96                        ;\
        #00F0=97                        ; |
        #0F00=98                        ; |
        #F000=99                        ; |
        #30000=9A                       ;  \  FREQUENTLY
        #7FFF8000=9B                    ;  /     USED DATA
        #540000=9C                      ; |
        #F00000=9D                      ; |
        #FC0000=9E                      ; |
        #3F003FFF=9F                    ;/
        #254=0A0                        ;POINTER TO MEMORY XFER DATA

        LB0=0B0                         ;TEMP LOCATION

        LD0=0D0                         ;TEMP LOCATION

        LF0=0F0                         ;TEMP LOCATION
.page "Microinstruction Formats"
;
;       The following is taken directly from the NEBULA Hardware
;       Specification.
;
;                       MICROINSTRUCTION FORMATS
;
;
;
;
;                22 21       16 15          9 8 7 6 5 4       0
;                +-----+-------+-------+-------+-------+-------+
;1. BASIC        |1|    DP    |    D ADRS   | B |CC |  SCTL   |
;                +-+-----------+-------------+---+---+---------+
;
;                22  20 19 17 16            9 8 7 6 5 4       0
;                +-----+-------+-------+-------+-------+-------+
;2. MOVE         |0 1 1| MDP |    XD ADRS    | B |CC |  SCTL   |
;                +-----+-----+---------------+---+---+---------+
;
;                22  20 19       14 13 11   9 8 7 6 5 4       0
;                +-----+-------+-------+-------+-------+-------+
;3. EXTENDED     |0 1 0|    XDP    | SI  | A | B |CC |  SCTL   |
;                +-----+-----------+-----+---+---+---+---------+
;
;
```

```
;               22    19 18 16 15          9 8 7 6 5 4       0
```

```
;
;                +-------+------+-------------+---+---+---------+
;4. MEM.REQ      |0 0 1 1| MF1  |   D ADRS    |MF2|DT |  SCTL   |
;                +-------+------+-------------+---+---+---------+
;
;                22    19 18 16 15   12 11  9 8 7 6 5 4        0
;                +-------+-+---+--------+-----+-+-+---+---------+
;5. MISC         |0 0 1 0|P|0 0|   M1   | M2  |0|R|0 0|  SCTL   |
;                +-------+-+---+--------+-----+-+-+---+---------+
;
;                22    19 18 17                       4 3      0
;                +-------+--+---------------------------+-------+
;6. JUMP         |0 0 0 1|OS|    JUMP ADDRESS           | JCTL  |
;                +-------+--+---------------------------+-------+
;
;                22    19 18 17      12 11  9 8 7 6 5 4 3      0
;                +-------+-+------------+-----+-+-+-+-+-+-------+
;7. DECODE       |0 0 0 0| | DEC.ADRS   |IFUNC| | | | | | JCTL  |
;                +-------+-+------------+-----+-+-+-+-+-+-------+
;                         |                    | | | | |
;                BACK UP PC           IB REQ---+ | | | +--OPC/SPEC
;                                                | | |
;                            SEL CM HI IR BYTE---+ | +--LOAD RDEST
;                                                  |
;                                         LOAD OS--+
;
.page
;
;1. "BASIC" Microinstruction
;
;    CSR<22> = 1      (OPCODE)
;
;        This opcode bit causes LS Adrs <7> to be cleared.
;
;    CSR<21:16>  (Data Path Control)
;
;        This field controls the data path. It controls the 2901 ALU,
;        the ALU data source, the data destination in the 2901 array,
;        and the write to the Local Store.
;
;    CSR<15:9>   (D Adrs <6:0>)
;
;        This field selects which of the 128 accessable Local Store
;        locations to use.
;
;    CSR<8:7>    (B Adrs)
;
;        This field selects one of the four Working Registers.
;
;    CSR<6:5>    (Condition Codes)
;
;        This field specifies the data type and condition code control.
;
;    CSR<4:0>    (SCTL)
;
;        This field specifies the skip condition/micro sequencer
;        function.
.page
```

```
;
```

```
;
;2. "MOVE" Microinstruction
;
;      CSR<22:20> = 011 (OPCODE)
;
;         This combination of opcode bits allows CSR<16> to control
;         LS Adrs <7>. This permits access of LS locations 80 - FF.
;
;      CSR<19:17> (Data Path Control)
;
;         This field is decoded to produce some data path control
;         information.
;
;           0 LS[XD] <- WR[B], WR[B] <- MEM DATA* 4 LS[XD] <- MEM DATA*
;           1 MEMORY <- LS [XD]*                  5 LS[XD] <- ACC/PORT
;           2 Q <- XWR[B]                         6 LS[XD] <- WR[B] + OS
;           3 WR[B] <- LS [XD]                    7 LS[XD] <- WR[B]
;
;      CSR<16:9>  (XD<7:0>)
;
;         This field specifies which of the 256 Local Store locations to
;         use.
;
;      CSR<8:7>   (B Adrs)
;
;         This field specifies which of the four Working Registers to
;         use. For MDP = 2, this field selects either the Backup PC or
;         the VAR.
;
;      CSR<6:5>   (CC)
;
;         This field specifies the data type and condition code control.
;
;      CSR<4:0>   (SCTL)
;
;         This field specifies the Skip control. See Table 3.
;
;
;* For information on which Working Registers and Local Store locations
;  may  be  used for Memory Addresses and data source/destinations, see
;  the Memory Management documentation.  Note that  these  restrictions
;  are due to microcoding conventions and not hardware.
.page
;
;
;3. "EXTENDED" Microinstruction
;
;     CSR<22:20> = 010    (OPCODE)
;
;         This opcode causes a function internal to the 2901 array; that
;         is, an operation involving only Working Registers and the Q-
;         Register.
;
;     CSR<19:14>  (Data Path Control)
;
;         This field is decoded to supply the 2901 ALU function code,
;         the ALU Source control, the destination code, and the ALU
```

```
;	Carry-In.
```

```
;
;    CSR<13:11>  (Shift Input)
;
;        This field selects the data to be shifted into a register,
;        when a Shift function is being done.
;
;    CSR<10:9>    (A Adrs)
;
;        This field selects which Working Register to use as a source
;        of data, when RAM A is selected to the ALU.
;
;    CSR<8:7>     (B Adrs)
;
;        This field selects which Working Register to use as a source
;        (when RAM B is selected to the ALU) and/or destination (when
;        the RAM is written) of data.
;
;    CSR<6:5>     (CC)
;
;        This field specifies the data type and condition code control.
;
;    CSR<4:0>     (SCTL)
;
;        This field specifies the skip condition/micro sequencer
;        function.
.page
;
;
;4. "MEM.REQ" Microinstruction
;
;    CSR<22:19> = 0011    (OPCODE)
;
;        This opcode causes a Memory Request to be started. The Local
;        Store is output on on the D-Bus, to be used as the virtual
;        address. Working Register 5 is written with this address.
;
;    CSR<18:16>, <8:7>   (Memory Function)
;
;        This field specifies to the Memory Controller the type of
;        request: physical, virtual, type of access, etc. For the
;        list of functions and the codes, see the NEBULA Memery Spec.
;
;    CSR<15:9>    (D Adrs <6:0>)
;
;        This field selects which of the 128 accessable Local Store
;        locations to use as the virtual address.
;
;    CSR<6:5>     (Data Type)
;
;        This field specifies the data type of the request.
;
;                00::=BYTE
;                01::=WORD
;                10::=SIZE Reg
;                11::=LONG
;
;    CSR<4:0>     (SCTL)
```

```
;
```

```
;       This field specifies the skip condition/micro sequencer
;       function.
.page
;
;
;5. "MISC" Microinstruction
;
;    CSR<22:19> = 0010 (OPCODE)
;
;       This combination of opcode bits causes the data path to no-op,
;       and enables the MISC decoders.
;
;    CSR<18>            (Port/Misc Select)
;
;       This bit defines the instruction to be either a Misc or a Port
;       instruction.  For CSR<18> = 1, CSR<17:10> is the Command Byte
;       to the Port, CSR<9:5> must be zero and CSR<4:0>  is  the  SCTL
;       field.  For CSR<18> = 0, it is a Misc instruction and the rest
;       of the word is defined as follows.
;
;    CSR<17:16>         (Not Used)
;
;    CSR<15:12>  (Misc Function 1)
;
;       These four bits are decoded to do the following functions:
;
;               0 = CLEAR State Reg<1:0>
;               1 = CLEAR State Reg<1>, SET State Reg<0>
;               2 = SET State Reg<1>, CLEAR State Reg<0>
;               3 = CLEAR State Reg<0>
;               4 = CLEAR State Reg<1>
;               5 = SET State Reg<0>
;               6 = SET State Reg<1>
;               7 = SET REGISTER BACKUP MASK FLAG
;               8 = SET ACC I/O MODE
;               9 = SET PORT I/O MODE
;               A = CLEAR WCS HI ADRS
;               B = SET WCS HI ADRS
;               C = CLEAR CPU ATTN AND CPU ACK
;               D = SET CPU ACK
;               E = SET CPU ATTN
;               F = NOP
;
;    CSR<11:9>   (Misc Function 2)
;
;       This field is decoded to do the following functions:
;
;               0 = NOP
;               1 = Mask IRD Interrupt Detection during Next State
;               2 = Assert CPU Data Available and pass WR to ACC/Port
;               3 = Trap ACC
;               4 = Read ACC uPC
;               5 = Assert XFER GRANT to Port
;               6 = Mask Halt and T-Bit Traps
;               7 = Write WR to Console Read Register
.page
;
```

```
;	CSR<8>		(MUST BE ZERO)
```

```
;
;    CSR<7>        (WR Address)
;
;
;        This bit selects WR[0] or WR[1] to be put on the Y-Bus.
;
;    CSR<6:5>               (Not Used)
;
;    CSR<4:0>     (SCTL)
;
;        This  field  specifies  the  skip  condition/micro   sequencer
;        function.
;
.page
;
;
;6. "JUMP" Microinstruction
;
;    CSR<22:19> = 0001     (OPCODE)
;
;        This opcode causes the data path to no-op, and the micro
;        sequencer to execute a JUMP.
;
;    CSR<18>       (Select OS)
;
;        This bit, when asserted, causes OS<4:0> to be OR'ed with the
;        five low bits of the of the jump microaddress.
;
;    CSR<17:4>     (Jump Microaddress)
;
;        If Select OS is CLEAR, this field specifies the fourteen-bit
;        jump micro address. If Select OS is SET, CSR<17:9> is Jump
;        Adrs<13:5> and OS<4:0> OR'ed with CSR<8:4> is Jump Adrs<4:0>.
;
;    CSR<3:0>      (JCTL)
;
;        This field specifies the jump condition/micro sequencer
;        function.
.page
;
;
;7. "DECODE" Microinstruction
;
;    A.  DESCRIPTION
;
;    CSR<22:19> = 0000 (OPCODE)
;
;        This combination of opcode bits  causes  the  Local  Store  to
;        access location 10 (the PC.) The Local Store drives the D-Bus.
;        The 2901 is set up to input the data on the  D-Bus,  increment
;        it,  and output to the Y-Bus.  Thus, the Y-Bus contains the PC
;        + 1.
;
;    CSR<18>       (BPC [asserted low])
;
;        If this bit is CLEAR, the incremented PC is written  into  the
;        2901 RAM.  If it is SET, the RAM is not written.
```

```
;
```

```
;    CSR<17:12>  (DEC.ADRS <13:8>)
;
;         This data is taken to be  the  upper  six  bits  of  the  next
;         microaddress.  The low eight bits are supplied by the IRD ROM.
;         The actual JUMP/JSR/NO-OP is determined by the JCTL field.
;
;    CSR<11:9>    (IFUNC)
;
;         This is the IFUNC field.   It  defines  which  fork  is  being
;         taken.
;
;    CSR<8>       (IB.REQ)
;
;         This is the IB.REQ bit.  When it  is  CLEAR,  the  LS  is  not
;         written,  no  interaction  with  memory  is  initiated and the
;         instructin buffer is not affected.
;
;         When it is SET:
;
;         The incremented PC (valid on the  Y-Bus  by  the  end  of  the
;         cycle)   is  re-written  to  Local   Store   location  0.   This
;         completes the PC increment function.
;
;         A Conditional Memory Request is executed.  If the two low bits
;         of  the  PC  are not 11, the request is not initiated.  If the
;         two  low  bits  are  set,  an  IB  PREFETCH  is  done.     The
;         (unincremented)  PC  is output to the memory and the CPU grant
;         (CPUG) signal is examined.  If CPUG is low by  T120,  the  CPU
;         will  stall  until  the  CPUG  signal becomes true.  After the
;         request is completed, the next state entered is dependent upon
;         the JCTL field.
.page
;
;    CSR<7>       (SEL CM HI IR BYTE)
;
;         This bit enables the upper  byte  of  the  Compatibility  Mode
;         instruction  in the Prefetch register to drive the IB-Bus.  It
;         is used only while in Compatibility Mode, when  IRD  is  being
;         done.  It must never be asserted in VAX Mode.
;
;
;    CSR<6>       (LD.OS)
;
;         If this bit is SET, the eight-bit data  on  the  IB  Bus  (the
;         output  of  the  instruction  buffer)  is  loaded  into the OS
;         register.  This load is dependent upon Compatibility Mode  and
;         LD  R  Dest  (If  CM.AND.LD R Dest, OS<3> <- 0.) If the bit is
;         CLEAR, OS is not affected.
;
;    CSR<5>       (LD R DEST)
;
;         If this bit is SET, the R  Dest  Skip  bit  will  be  SET  for
;         Specifier Mode equal to 5 and CLEARED for Mode not equal to 5,
;         in VAX mode, or mode 0 while in Compatibility Mode.   If  this
;         bit is CLEAR, the R Dest bit will be unaffected.
;
;    CSR<4>              (OPC/SPEC)
```

```
;
```

```
;        This bit (effectively another IFUNC bit) defines the  data  to
;        be  decoded;  that is, if it is an Opcode or a Specifier.  In
;        hardware, it selects which ROM is to drive the  micro  address
;        bus.
;
;    CSR<3:0>     (JCTL)
;
;        These four bits are decoded to make one of 16 functions.   See
;        Table 3.
.page "Tables"
;
;                              TABLES
;
;
;
;                 Table 1.  2901 Control Decoding
;
;
;        The 2901 control decoder examines CSR<22:14>. This nine-bit
;        field  of  the microword changes in meaning for the different
;        cases of micro opcode.
;
;        CSR      22  21  20  19  18  17  16  15  14
;                 --  --  --  --  --  --  --  --  --
;1. BASIC          1  [Six bit function code]  X   X
;2. MOVE           0   1   1  [function]   X   X   X
;3. EXTENDED       0   1   0  [Six bit function code]
;4. MEM.REQ        0   0   1   1   X   X   X   X   X     WR[5] <- D
;5. MISC           0   0   1   0   X   X   X   X   X     Y-Bus <- WR
;6. JUMP           0   0   0   1   X   X   X   X   X     NO-OP
;7. DECODE         0   0   0   0  BPC* X   X   X   X
;
;  If BPC = 0, Write WR; else, No-op
;
;      This decoder is implemented using a ROM and a PAL. 512
;      locations are needed for CSR<22:14> to index into. The
;      decoding logic supplies 10 control bits:
;
;             ALU Source Select   (3)
;             ALU Function   (3)
;             ALU Result Destination   (3)
;             ALU Carry-In   (1)
;
;      The ROM addresses are therefore allocated in the following way:
;
;      Location      0 -   F       Incr. D-Bus, Output ALU, No Write.
;                   10 -  1F       Incr. D-Bus, Output ALU, Write RAM.
;                   20 -  3F       No-Op.
;                   40 -  5F       Output WR, bypassing ALU.
;                   60 -  7F       Pass D-Bus through ALU and write RAM.
;                   80 -  BF       CSR<19:14> indicates 2901 function.
;                   C0 -  FF       CSR<19:17> indicates 2901 function.
;                  100 - 1FF       CSR<21:16> indicates 2901 function.
;
;      For the detailed function table see the Microcode Listing.
.page
;
```

```
;
```

```
;                       Table 2. Local Store Write
;
;       The Local Store Write Controller examines CSR<22:17>, CSR<6>, the
;       SIZE  register  (SIZE<1:0>)  and the Compatibility Mode bit (CM).
;       These eight bits are decoded to determine which parts of  the  LS
;       to  write, if any.  The LS is written with data from the 2901 ALU
;       (Y-Bus).
;
;                 CSR      22  21  20  19  18  17
;                          --  --  --  --  --  --
;           1. BASIC        1   0   X   X   X   X    No Write
;                           1   1   X   X   X   X    Write *
;           2. MOVE         0   1   1   0   X   1    No Write
;                           0   1   1   0   1   X    No Write
;                           0   1   1   1   X   X    Write *
;           3. EXTENDED     0   1   1   X   X   X    No Write
;           4. MEM.REQ      0   0   1   1   X   X    No Write
;           5. MISC         0   0   1   0   X   X    No Write
;           6. JUMP         0   0   0   1   X   X    No Write
;           7. DECODE       0   0   0   0   X   X    Conditional Write **
;
;    * LS Write, Data Type Dependent. These cases now examine CSR<6>.
;
;      If CSR<6> = 0, and CM = 0   Write LS<31:0>  (Data Type = LONG)
;      If CSR<6> = 0, and CM = 1   Write LS<15:0>  (Compatibility
;                                                   Mode. DT = WORD)
;
;      If CSR<6> = 1, the Write is conditional on the SIZE register.
;
;      If CM = 0   SIZE<1:0> = 00  Write LS<7:0>  (BYTE)
;                            = 01  Write LS<15:0> (WORD)
;                            = 1X  Write LS<31:0> (LONG)
;
;      If CM = 1  Write LS<15:0>  (WORD)
;
;    ** If CSR<8> = 0, No Write
;       If CSR<8> = 1, Write LS<31:0>  (Writes incremented PC)
.page
;
;
;                 Table 3. SKIP/JUMP Conditions
;
;This table summarizes the Skip and Jump conditions, and the other
;special functions of the micro sequencer.
;
;SCTL codes 0-F, 17-1F are Skip conditions. The codes 10-17 execute
;special functions. JCTL codes 0-B are Jump conditions, and C-F
;execute special functions.
;
;
;Sctl     Function                  Sctl     Function
;----     --------                  ----     --------
; 00      Not ALU N                  10      RTN+1 if No Mem Err
; 01      Not ALU Z                  11      Loop if Not ALU Z
; 02      Not ALU V                  12      POP Stack
; 03      ALU C                      13      Loop if ALU C
; 04      Not PSL C                  14      Return
; 05	Branch False		15	No Skip (NOP)
; 06      RDEST                       16      RTN+1
; 07      Not Inter. Req.             17      Loop if No Inter. and No Sync
; 08      ALU N                       18      Console ATTN
; 09      ALU Z                       19      Console ACK
; 0A      ALU V                       1A      Port Interrupt Pending
; 0B      Not ALU C                   1B      Reg. Backup Mask Flag
; 0C      State 0                     1C      ALU N.XOR.ALU V
; 0D      State 1                     1D      Not Memory Error
; 0E      Not State 0                 1E      Skip
; 0F      Not State 1                 1F      Sync
;
;
;Jctl     Function                   Jctl     Function
;----     --------                   ----     --------
; 00      Not ALU N                   08      ALU N
; 01      Not ALU Z                   09      ALU Z
; 02      Not ALU V                   0A      ALU V
; 03      ALU C                       0B      Not ALU C
; 04      JUMP                        0C      JSR
; 05      Branch False                0D      JSR if IB Valid
; 06      RDEST                       0E      No Jump; Skip if IB Valid
; 07      Not Interr. Req             0F      IB Valid
;

.PAGE   "2901 ALU Control Description"

;
;               The 2901 ALU is the heart  of  the  CPU  module.   It  is
;          responsible for all CPU calculations.  The control lines going
;          to I0 through I8 control the function, input,  and  output  of
;          the  2901.   When  a  failure  involving  the 2901 occurs, the
;          logical place to start looking  is  the  control  lines.   The
;          following  tables  describe what the control signals should be
;          for various functions.  Use it whenever the control lines  are
;          to  be  checked.  The test descriptions tell the operator what
;          function the 2901 is supposed to be doing.
;
;               The SRC CTL on lines I0 through I2 (pins  12  through  14
;          respectively)  control  where  the  inputs to the internal ALU
;          within the 2901 come from.  The inputs to the internal ALU are
;          labeled R and S.  R is one operand (4 bits) and S is the other
;          operand (also 4 bits).  The R input can come  from  a  Working
;          Register  (labeled  A), the D bus (direct input), or be forced
;          to 0.  The S  input  can  come  from  either  set  of  working
;          registers  (labeled  A and B), the Q register, or forced to 0.
;          A table of these control signals follow:
;
;
;          ----------------------------
;          |                |     | SRC   |
;          |                |     |operands|
;          |                |octal|--------|
;          | I2   I1   I0   |code |  R  S  |
;          |---------------------------------
;          | L    L    L    |  0  |  A  Q  |
;          | L    L    H    |  1  |  A  B  |
;          | L    H    L    |  2  |  0  Q  |
;               |L   H   H   | 3 | 0  B |
;               | H    L    L    |  4  |  0  A |
;               | H    L    H    |  5  |  D  A |
;               | H    H    L    |  6  |  D  Q |
;               | H    H    H    |  7  |  D  0 |
;               ---------------------------------
;
;                 ALU SOURCE OPERAND CONTROL
;
;
;
;
;
;                  The ALU CTL on lines I3  through  I5  (pins  26,  28,  27
;               respectively)  control  what function the ALU is to perform on
;               the two operands R and S.  A table follows:
.page
;
;               ------------------------------------
;               |               |octal|     ALU     |
;               | I5   I4   I3  |code |  function   |
;               ------------------------------------
;               | L    L    L   |  0  |R PLUS S     |
;               | L    L    H   |  1  |S MINUS R    |
;               | L    H    L   |  2  |R MINUS S    |
;               | L    H    H   |  3  |R OR S       |
;               | H    L    L   |  4  |R AND S      |
;               | H    L    H   |  5  |R NOT AND S  |
;               | H    H    L   |  6  |R XOR S      |
;               | H    H    H   |  7  |R XNOR S     |
;               ------------------------------------
;                    ALU FUNCTION CONTROL
;
;
;
;
;                  The DST CTL  on  lines  I6  through  I8  (pins  5,  7,  6
;               respectively) control what goes out onto the Y bus (either the
;               internal ALU  result  or  a  WR  directly)  and  what  happens
;               internal  to  the 2901 (shifting, writing the Q register etc).
;               Internal shifting left or  right  is  accomplished  with  this
;               control.   Below is a table explaining the various destination
;               possibilities.  The arrow refers to where the output will  end
;               up.   B  indicates the WR addressed by the B ADDR, F refers to
;               the output of the internal ALU, and A refers to the output  of
;               the  WR  addressed by the A ADDR.  Up is toward the MSB, while
;               down is toward the LSB.
;
;
;       ---------------------------------------------------------------------------------
;       |nomenclature|                |octal|  RAM function   |  Q-reg function|   Y    |
;       | in listing |  I8   I7   I6  |code |  shift  | load  |  shift | load  | output |
;       ---------------------------------------------------------------------------------
;       | LOAD Q     |   L    L    l  |  0  |  NONE   | NONE  |  NONE  | F->Q  |   F    |
;       |            |                |     |         |       |        |       |        |
;       | NOP        |   L    L    H  |  1  |  NONE   | NONE  |  NONE  | NONE  |   F    |
;       |            |                |     |         |       |        |       |        |
;       | WRITE.B.A  |   L    H    L  |  2  |  NONE   | F->B  |  NONE  | NONE  |   A    |
;       |            |                |     |         |       |        |       |        |
;       | WRITE.B.F  |   L    H    H  |  3  |  NONE   | F->B  |  NONE  | NONE  |   F    |
; |                |                |      |         |         |         |         |         |
;   | RSHF.RAM.Q |  H   L   L |  4  | DOWN  | F/2->B| DOWN  | Q/2->Q|   F   |
;   |            |            |     |       |       |       |       |       |
;   | RSHF.RAM   |  H   L   H |  5  | DOWN  | F/2->B| NONE  | NONE  |   F   |
;   |            |            |     |       |       |       |       |       |
;   | LSHF.RAM.Q |  H   H   L |  6  |  UP   | 2F->B |  UP   | 2Q->Q |   F   |
;   |            |            |     |       |       |       |       |       |
;   | LSHF.RAM   |  H   H   H |  7  |  UP   | 2F->B | NONE  | NONE  |   F   |
;    -----------------------------------------------------------------------
;                         ALU DESTINATION CONTROL
;
.page
.TOC     "DIAGNOSTIC MACROS"

LS.DATA.WORD/=<15:0>
UPPER.BYTE/=<23:16>

WORD[]              "UPPER.BYTE/0,LS.DATA.WORD/@1"
COUNT.LS[]          "UPPER.BYTE/0,LS.DATA.WORD/@1"
COUNT.MM[]          "UPPER.BYTE/1,LS.DATA.WORD/@1"
START.ADR[]         "UPPER.BYTE/0,LS.DATA.WORD/@1"

ASCII.LIT/=<15:0>

        CA=4341         ;ASCII VALUE FOR FILE NAMES
        CB=4342         ;  !
        CC=4343         ;  !
        CD=4344         ;  !
        CE=4345         ;  V
        CF=4346
        CG=4347
        CH=4348

ASCII []            "UPPER.BYTE/0,ASCII.LIT/@1"

.REGION/0,6F
.PAGE    "TEST POINTERS"
;
;       This portion contains the pointers to all tests in this section.  The
;       WCS address of the JUMP instruction corresponds to the test number.
;       E.g. WCS 5 contains a JUMP to test 5.  All unused test numbers will
;       contain a JUMP to an error routine.  This portion is 63. locations
;       long, allowing for up to 63. tests per section, from WCS address 1 to
;       WCS address 3F.
;
1:                                      ;TEST POINTERS BEGIN AT WCS ADDRESS 1

        JMP [T.1]                       ;GO TO TEST 1
        JMP [T.2]                       ;GO TO TEST 2
        JMP [T.3]                       ;GO TO TEST 3
        JMP [T.4]                       ;GO TO TEST 4
        JMP [T.5]                       ;GO TO TEST 5
        JMP [T.6]                       ;GO TO TEST 6
        JMP [T.7]                       ;GO TO TEST 7
        JMP [T.8]                       ;GO TO TEST 8
        JMP [T.9]                       ;GO TO TEST 9
        JMP [T.A]                       ;GO TO TEST A
        JMP [T.B]                       ;GO TO TEST B
```

```
	JMP [T.C]		;GO TO TEST C
```

```
        JMP [T.D]                       ;GO TO TEST D
        JMP [T.E]                       ;GO TO TEST E
        JMP [T.F]                       ;GO TO TEST F
        JMP [T.10]                      ;GO TO TEST 10
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
```

```
                        ; IN THIS OVERLAY
```

```
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                        ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]               ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
```

```
                                        ; IN THIS OVERLAY
```

```
        JMP [ERR.NO.TEST]                       ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                                ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]                       ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                                ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]                       ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                                ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]                       ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                                ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]                       ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                                ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]                       ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                                ; IN THIS OVERLAY
        JMP [ERR.NO.TEST]                       ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                                                ; IN THIS OVERLAY
.PAGE   "DIAGNOSTIC POINTERS"
;
;       This section contains all the pointers needed for this diagnostic
;       overlay.  The first pointer (at WCS location 0) points to a data
;       transfer routine.  It is the first location executed after a new
;       WCS load.  The insruction here is a JUMP to TRANSFER.POINT which may
;       contain instructions to transfer data.  If no data is to be trans-
;       ferred, or at the completion of the data transfers, the CPU will
;       issue CPU ATTN with all bits of the control and status word clear.
;
;       The next 80. locations (from WCS location 1 to WCS location 4F)
;       contain pointers to each test in this overlay.  The pointers are
;       JUMP instructions to the test number corresponding to the location
;       number.  E.g. location 5 will contain a JUMP to test 5.  All unused
;       test numbers will contain a JUMP to an error routine.  This portion
;       appears after the definitions in this listing.
;
;       Finally the next 32. locations (from WCS location 50 to 6F) contain
;       pointers (Jump instructions) to WCS subroutines which are to be used by
;       the console diagnostic monitor.
;
0:      JMP [TRANSFER.POINT]                    ;GO TO ROUTINE THAT LOADS LS 7 WITH
                                                ; POINTER TO DATA SECTION AND ISSUES
                                                ; CPU ATTN WITH TRANSFER BIT SET.

50:                                             ;START AT WCS LOCATION 50 FOR SUB-
                                                ; ROUTINE POINTERS
;
; SUBROUTINE POINTERS
;
        JMP [DEPOSIT.CSR]                       ;GO TO WCS SUBROUTINE WHICH WRITES THE
                                                ; MCT CSR FOR DEPOSIT CSR COMMAND
        JMP [EXAMINE.CSR]                       ;GO TO WCS SUBROUTINE WHICH READS THE
                                                ; MCT CSR FOR EXAMINE CSR COMMAND
        JMP [DEPOSIT.TB]                        ;GO TO WCS SUBROUTINE WHICH WRITES THE
                                                ; MCT TRANSLATION BUFFER FOR DEPOSIT
                                                ; TB COMMAND
        JMP [EXAMINE.TB]                        ;GO TO WCS SUBROUTINE WHICH READS THE
                                                ; MCT TRANSLATION BUFFER FOR EXAMINE
                                                ; TB COMMAND
        JMP [DEPOSIT.MM]                        ;GO TO WCS SUBROUTINE WHICH WRITES MAIN
                                                ; MEMORY FOR DEPOSIT MM COMMAND.  ALSO
                                                ; USED TRANSFERING DATA FROM WCS TO
; MAIN MEMORY.
```

```
        JMP [EXAMINE.MM]                ;GO TO WCS SUBROUTINE WHICH READS MAIN
                                        ; MEMORY FOR EXAMINE MM COMMAND.
        JMP [SAVE.WR]                   ;GO TO WCS SUBROUTINE WHICH STORES THE
                                        ; CONTENTS OF WR0-WR3 IN LS 0-3
        JMP [RESTORE.WR]                ;GO TO WCS SUBROUTINE WHICH RESTORES THE
                                        ; CONTENTS OF WR0-WR3 FROM LS 0-3
        JMP [DEPOSIT.UBS]               ;GO TO WCS SUBROUTINE WHICH WRITES THE
                                        ; UNIBUS MAP ON THE MEMORY CONTROLLER
        JMP [EXAMINE.UBS]               ;GO TO WCS SUBROUTINE WHICH READS THE
                                        ; UNIBUS MAP ON THE MEMORY CONTROLLER
        JMP [EXAMINE.ICSR]              ;GO TO WCS SUBROUTINE WHICH READS THE
                                        ; IDC CSR
        JMP [EXAMINE.IDAR]              ;GO TO WCS SUBROUTINE WHICH READS THE
                                        ; IDC DAR
        JMP [EXAMINE.DBUF]              ;GO TO WCS SUBROUTINE WHICH READS THE
                                        ; IDC DBUF
        JMP [EXAMINE.PATT]              ;GO TO WCS SUBROUTINE WHICH READS THE
                                        ; IDC ECC PATTERN
        JMP [EXAMINE.POSIT]             ;GO TO WCS SUBROUTINE WHICH READS THE
                                        ; IDC ECC POSITION
        JMP [DEPOSIT.ICSR]              ;GO TO WCS SUBROUTINE WHICH WRITES THE
                                        ; IDC CSR
        JMP [DEPOSIT.IDAR]              ;GO TO WCS SUBROUTINE WHICH WRITES THE
                                        ; IDC DAR
        JMP [DEPOSIT.DBUF]              ;GO TO WCS SUBROUTINE WHICH WRITES THE
                                        ; IDC DBUF
        JMP [CLEAR.FIFO.ADDR]           ;GO TO THE WCS SUBROUTINE WHICH CLEARS
                                        ; THE IDC FIFO ADDRESS
        JMP [SELECT.FIFO.A]             ;GO TO THE WCS SUBROUTINE WHICH SELECTS
                                        ; FIFO A
        JMP [SELECT.FIFO.B]             ;GO TO THE WCS SUBROUTINE WHICH SELECTS
                                        ; FIFO B
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
        NOP                             ;NOT USED
.PAGE
.REGION/70,5FF
.TOC    "LS DATA SECTION"

;+++
;       This section lists the data that will be transfered to LS by the con-
;       sole as part of the initialization performed in test 0.  The TRANSFER
;       DATA routine will point to this data area.  LS contains the constants,
;       control and status word, and temporary storage locations used by the
;       tests and subroutines of the microdiagnostic.
;         The LS is 32 bits wide.  Each of these words is 16 bits, so two con-
;       secutive words listed here will be loaded into each consecutive LS loc-
;       ation.  The first word listed will be bits 0-15 of the LS location, the
;       second word listed will be bits 16-31 of the same LS location.
;       The locations 78-7F in the first half of LS and F8-FF in the second
;       half of LS are not actual LS locations.  They force an access to one of
;       several registers in the CPU or force an indexing feature to another LS
;       location.  For that reason, they are not written via the transfer rou-
;       tine.
;         Note:  This table is in hexadecimal format i.e. each digit represents
;       four bits, so four digits represnts a full 16 bit word.  The micro-
;       assembler requires that a hexidecimal value that starts with a letter
;       (e.g. FFFFF) be preceeded by a 0 (0FFFF).  So some table values will
;       contain 5 hexidecimal digits.  The fifth digit is not actually used.
;
;---

TRANSFER.DATA.LS1:

        COUNT.LS[0078]          ;LONGWORD COUNT (NUMBER OF LONGWORDS TO TRANS-
                                ; FER TO LS = 120 DECIMAL) DESTINATION IS LS
        START.ADR[0000]         ;DESTINATION START ADDRESS (START AT LS 0)

        WORD[0000]              ;LOCATION 0
        WORD[0000]

        WORD[0000]              ;LOCATION 1
        WORD[0000]

        WORD[0000]              ;LOCATION 2
        WORD[0000]

        WORD[0000]              ;LOCATION 3
        WORD[0000]

        WORD[0000]              ;LOCATION 4
        WORD[0000]

        WORD[0000]              ;LOCATION 5
        WORD[0000]

        WORD[0000]              ;LOCATION 6
        WORD[0000]

        WORD[0000]              ;LOCATION 7
        WORD[0000]

        WORD[0000]              ;LOCATION 8
        WORD[0000]

        WORD[0000]              ;LOCATION 9
        WORD[0000]

        WORD[0000]              ;LOCATION 0A
        WORD[0000]

        WORD[0000]              ;LOCATION 0B
        WORD[0000]

        WORD[0000]              ;LOCATION 0C
        WORD[0000]
        WORD[0000]                  ;LOCATION 0D
        WORD[0000]

        WORD[0000]                  ;LOCATION 0E
        WORD[0000]

        WORD[0000]                  ;LOCATION 0F
        WORD[0000]

        WORD[0000]                  ;LOCATION 10
        WORD[0000]

        WORD[0000]                  ;LOCATION 11
        WORD[0000]

        WORD[0000]                  ;LOCATION 12
        WORD[0000]

        WORD[00FF]                  ;LOCATION 13
        WORD[0000]

        WORD[0FFFF]                 ;LOCATION 14
        WORD[0000]

        WORD[0000]                  ;LOCATION 15
        WORD[0FF00]

        WORD[0FF00]                 ;LOCATION 16
        WORD[0FFFF]

        WORD[3FFF]                  ;LOCATION 17
        WORD[0100]

        WORD[3FFF]                  ;LOCATION 18
        WORD[0FFFE]

        WORD[0FFFF]                 ;LOCATION 19
        WORD[0FE7F]

        WORD[0000]                  ;LOCATION 1A
        WORD[7FF8]

        WORD[8000]                  ;LOCATION 1B
        WORD[7FFF]

        WORD[0000]                  ;LOCATION 1C
        WORD[0000]

        WORD[0000]                  ;LOCATION 1D
        WORD[0000]

        WORD[0500]                  ;LOCATION 1E
        WORD[0080]

        WORD[0500]                  ;LOCATION 1F
        WORD[0000]

        WORD[0001]                  ;LOCATION 20
```

```
WORD[0000]
```

```
        WORD[0002]              ;LOCATION 21
        WORD[0000]

        WORD[0004]              ;LOCATION 22
        WORD[0000]

        WORD[0008]              ;LOCATION 23
        WORD[0000]

        WORD[0010]              ;LOCATION 24
        WORD[0000]

        WORD[0020]              ;LOCATION 25
        WORD[0000]

        WORD[0040]              ;LOCATION 26
        WORD[0000]

        WORD[0080]              ;LOCATION 27
        WORD[0000]

        WORD[0100]              ;LOCATION 28
        WORD[0000]

        WORD[0200]              ;LOCATION 29
        WORD[0000]

        WORD[0400]              ;LOCATION 2A
        WORD[0000]

        WORD[0800]              ;LOCATION 2B
        WORD[0000]

        WORD[1000]              ;LOCATION 2C
        WORD[0000]

        WORD[2000]              ;LOCATION 2D
        WORD[0000]

        WORD[4000]              ;LOCATION 2E
        WORD[0000]

        WORD[8000]              ;LOCATION 2F
        WORD[0000]

        WORD[0000]              ;LOCATION 30
        WORD[0001]

        WORD[0000]              ;LOCATION 31
        WORD[0002]

        WORD[0000]              ;LOCATION 32
        WORD[0004]

        WORD[0000]              ;LOCATION 33
        WORD[0008]
```

```
        WORD[0000]              ;LOCATION 34
        WORD[0010]

        WORD[0000]              ;LOCATION 35
        WORD[0020]

        WORD[0000]              ;LOCATION 36
        WORD[0040]

        WORD[0000]              ;LOCATION 37
        WORD[0080]

        WORD[0000]              ;LOCATION 38
        WORD[0100]

        WORD[0000]              ;LOCATION 39
        WORD[0200]

        WORD[0000]              ;LOCATION 3A
        WORD[0400]

        WORD[0000]              ;LOCATION 3B
        WORD[0800]

        WORD[0000]              ;LOCATION 3C
        WORD[1000]

        WORD[0000]              ;LOCATION 3D
        WORD[2000]

        WORD[0000]              ;LOCATION 3E
        WORD[4000]

        WORD[0000]              ;LOCATION 3F
        WORD[8000]

        WORD[0000]              ;LOCATION 40
        WORD[0000]

        WORD[0000]              ;LOCATION 41
        WORD[0000]

        WORD[0000]              ;LOCATION 42
        WORD[0000]

        WORD[0000]              ;LOCATION 43
        WORD[0000]

        WORD[0000]              ;LOCATION 44
        WORD[0000]

        WORD[0000]              ;LOCATION 45
        WORD[0000]

        WORD[0000]              ;LOCATION 46
        WORD[0000]
```

```
	WORD[0000]		;LOCATION 47
```

```
        WORD[0000]

        WORD[0000]              ;LOCATION 48
        WORD[0000]

        WORD[0000]              ;LOCATION 49
        WORD[0000]

        WORD[0000]              ;LOCATION 4A
        WORD[0000]

        WORD[5555]              ;LOCATION 4B
        WORD[0AAAA]

        WORD[0AAAA]             ;LOCATION 4C
        WORD[0AAAA]

        WORD[5555]              ;LOCATION 4D
        WORD[5555]

        WORD[0000]              ;LOCATION 4E
        WORD[0000]

        WORD[0FFFF]             ;LOCATION 4F
        WORD[0FFFF]

        WORD[0000]              ;LOCATION 50
        WORD[0000]

        WORD[0000]              ;LOCATION 51
        WORD[0000]

        WORD[0000]              ;LOCATION 52
        WORD[0000]

        WORD[0000]              ;LOCATION 53
        WORD[0000]

        WORD[0000]              ;LOCATION 54
        WORD[0000]

        WORD[0000]              ;LOCATION 55
        WORD[0000]

        WORD[0000]              ;LOCATION 56
        WORD[0000]

        WORD[0000]              ;LOCATION 57
        WORD[0000]

        WORD[0000]              ;LOCATION 58
        WORD[0000]

        WORD[0000]              ;LOCATION 59
        WORD[0000]

        WORD[0000]              ;LOCATION 5A
        WORD[0000]
        WORD[0000]              ;LOCATION 5B
        WORD[0000]

        WORD[0000]              ;LOCATION 5C
        WORD[0000]

        WORD[0000]              ;LOCATION 5D
        WORD[0000]

        WORD[0000]              ;LOCATION 5E
        WORD[0000]

        WORD[0000]              ;LOCATION 5F
        WORD[0000]

        WORD[0003]              ;LOCATION 60
        WORD[0000]

        WORD[0012]              ;LOCATION 61
        WORD[0000]

        WORD[3333]              ;LOCATION 62
        WORD[3333]

        WORD[0F0F]              ;LOCATION 63
        WORD[0F0F]

        WORD[00FF]              ;LOCATION 64
        WORD[00FF]

        WORD[0162]              ;LOCATION 65
        WORD[0000]

        WORD[0000]              ;LOCATION 66
        WORD[0000]

        WORD[0000]              ;LOCATION 67
        WORD[0000]

        WORD[0000]              ;LOCATION 68
        WORD[0000]

        WORD[0000]              ;LOCATION 69
        WORD[0000]

        WORD[0000]              ;LOCATION 6A
        WORD[0000]

        WORD[0000]              ;LOCATION 6B
        WORD[0000]

        WORD[0000]              ;LOCATION 6C
        WORD[0000]

        WORD[0000]              ;LOCATION 6D
        WORD[0000]
```

```
        WORD[0000]                 ;LOCATION 6E
        WORD[0000]

        WORD[0000]                 ;LOCATION 6F
        WORD[0000]

        WORD[0000]                 ;LOCATION 70
        WORD[0000]

        WORD[0000]                 ;LOCATION 71
        WORD[0000]

        WORD[0000]                 ;LOCATION 72
        WORD[0000]

        WORD[0000]                 ;LOCATION 73
        WORD[0000]

        WORD[0000]                 ;LOCATION 74
        WORD[0000]

        WORD[0000]                 ;LOCATION 75
        WORD[0000]

        WORD[0000]                 ;LOCATION 76
        WORD[0000]

        WORD[0000]                 ;LOCATION 77
        WORD[0000]

.TOC    "PROGRAM SPECIFIC DATA SECTION (LS 80-F7)"
;
;       This section represents the second half of LS.  It is only acceseble
;       with a MOV instruction, or one that uses the MOVE microinstruction for-
;       mat.  This section will be loaded in a seperate transfer.
;
TRANSFER.DATA.LS2:
        COUNT.LS[0078]             ;LONGWORD COUNT (NUMBER OF LONGWORDS TO TRANS-
                                   ; FER TO LS = 120 DECIMAL) DESTINATION IS LS
        START.ADR[0080]            ;DESTINATION START ADDRESS (START AT LS 80(H))

        WORD[003C]                 ;LOCATION 80
        WORD[0000]

        WORD[0043]                 ;LOCATION 81
        WORD[0000]

        WORD[0064]                 ;LOCATION 82
        WORD[0000]

        WORD[0025]                 ;LOCATION 83
        WORD[0000]

        WORD[0026]                 ;LOCATION 84
        WORD[0000]

        WORD[0020]                 ;LOCATION 85
```

```
    WORD[0000]
```

```
WORD[0054]              ;LOCATION 86
WORD[0000]

WORD[007D]              ;LOCATION 87
WORD[0000]

WORD[0000]              ;LOCATION 88
WORD[0000]

WORD[0000]              ;LOCATION 89
WORD[0000]

WORD[0000]              ;LOCATION 8A
WORD[0000]

WORD[0000]              ;LOCATION 8B
WORD[0000]

WORD[0000]              ;LOCATION 8C
WORD[0000]

WORD[0000]              ;LOCATION 8D
WORD[0000]

WORD[0000]              ;LOCATION 8E
WORD[0000]

WORD[0000]              ;LOCATION 8F
WORD[0000]

WORD[0000]              ;LOCATION 90
WORD[0000]

WORD[5555]              ;LOCATION 91
WORD[0AAAA]

WORD[0000]              ;LOCATION 92
WORD[01F0]

WORD[0000]              ;LOCATION 93
WORD[0000]

WORD[0000]              ;LOCATION 94
WORD[0000]

WORD[0FFFC]             ;LOCATION 95
WORD[0FFFF]

WORD[000F]              ;LOCATION 96
WORD[0000]

WORD[00F0]              ;LOCATION 97
WORD[0000]

WORD[0F00]              ;LOCATION 98
WORD[0000]
```

```
WORD[0F000]             ;LOCATION 99
WORD[0000]

WORD[0000]              ;LOCATION 9A
WORD[0003]

WORD[8000]              ;LOCATION 9B
WORD[7FFF]

WORD[0000]              ;LOCATION 9C
WORD[0054]

WORD[0000]              ;LOCATION 9D
WORD[00F0]

WORD[0000]              ;LOCATION 9E
WORD[00FC]

WORD[3FFF]              ;LOCATION 9F
WORD[3F00]

WORD[0254]              ;LOCATION 0A0
WORD[0000]

WORD[0000]              ;LOCATION 0A1
WORD[0000]

WORD[0000]              ;LOCATION 0A2
WORD[0000]

WORD[0000]              ;LOCATION 0A3
WORD[0000]

WORD[0000]              ;LOCATION 0A4
WORD[0000]

WORD[0000]              ;LOCATION 0A5
WORD[0000]

WORD[0000]              ;LOCATION 0A6
WORD[0000]

WORD[0000]              ;LOCATION 0A7
WORD[0000]

WORD[0000]              ;LOCATION 0A8
WORD[0000]

WORD[0000]              ;LOCATION 0A9
WORD[0000]

WORD[0000]              ;LOCATION 0AA
WORD[0000]

WORD[0000]              ;LOCATION 0AB
WORD[0000]
```

```
	WORD[0000]		;LOCATION 0AC
```

```
        WORD[0000]

        WORD[0000]                  ;LOCATION 0AD
        WORD[0000]

        WORD[0000]                  ;LOCATION 0AE
        WORD[0000]

        WORD[0000]                  ;LOCATION 0AF
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B0
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B1
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B2
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B3
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B4
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B5
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B6
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B7
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B8
        WORD[0000]

        WORD[0000]                  ;LOCATION 0B9
        WORD[0000]

        WORD[0000]                  ;LOCATION 0BA
        WORD[0000]

        WORD[0000]                  ;LOCATION 0BB
        WORD[0000]

        WORD[0000]                  ;LOCATION 0BC
        WORD[0000]

        WORD[0000]                  ;LOCATION 0BD
        WORD[0000]

        WORD[0000]                  ;LOCATION 0BE
        WORD[0000]

        WORD[0000]                  ;LOCATION 0BF
```

```
WORD[0000]
```

```
        WORD[0000]                    ;LOCATION 0C0
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C1
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C2
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C3
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C4
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C5
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C6
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C7
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C8
        WORD[0000]

        WORD[0000]                    ;LOCATION 0C9
        WORD[0000]

        WORD[0000]                    ;LOCATION 0CA
        WORD[0000]

        WORD[0000]                    ;LOCATION 0CB
        WORD[0000]

        WORD[0000]                    ;LOCATION 0CC
        WORD[0000]

        WORD[0000]                    ;LOCATION 0CD
        WORD[0000]

        WORD[0000]                    ;LOCATION 0CE
        WORD[0000]

        WORD[0000]                    ;LOCATION 0CF
        WORD[0000]

        WORD[0000]                    ;LOCATION 0D0
        WORD[0000]

        WORD[0000]                    ;LOCATION 0D1
        WORD[0000]

        WORD[0000]                    ;LOCATION 0D2
        WORD[0000]
```

```
        WORD[0000]              ;LOCATION 0D3
        WORD[0000]

        WORD[0000]              ;LOCATION 0D4
        WORD[0000]

        WORD[0000]              ;LOCATION 0D5
        WORD[0000]

        WORD[0000]              ;LOCATION 0D6
        WORD[0000]

        WORD[0000]              ;LOCATION 0D7
        WORD[0000]

        WORD[0000]              ;LOCATION 0D8
        WORD[0000]

        WORD[0000]              ;LOCATION 0D9
        WORD[0000]

        WORD[0000]              ;LOCATION 0DA
        WORD[0000]

        WORD[0000]              ;LOCATION 0DB
        WORD[0000]

        WORD[0000]              ;LOCATION 0DC
        WORD[0000]

        WORD[0000]              ;LOCATION 0DD
        WORD[0000]

        WORD[0000]              ;LOCATION 0DE
        WORD[0000]

        WORD[0000]              ;LOCATION 0DF
        WORD[0000]

        WORD[0000]              ;LOCATION 0E0
        WORD[0000]

        WORD[0000]              ;LOCATION 0E1
        WORD[0000]

        WORD[0000]              ;LOCATION 0E2
        WORD[0000]

        WORD[0000]              ;LOCATION 0E3
        WORD[0000]

        WORD[0000]              ;LOCATION 0E4
        WORD[0000]

        WORD[0000]              ;LOCATION 0E5
        WORD[0000]
        WORD[0000]                  ;LOCATION 0E6
        WORD[0000]

        WORD[0000]              ;LOCATION 0E7
        WORD[0000]

        WORD[0000]              ;LOCATION 0E8
        WORD[0000]

        WORD[0000]              ;LOCATION 0E9
        WORD[0000]

        WORD[0000]              ;LOCATION 0EA
        WORD[0000]

        WORD[0000]              ;LOCATION 0EB
        WORD[0000]

        WORD[0000]              ;LOCATION 0EC
        WORD[0000]

        WORD[0000]              ;LOCATION 0ED
        WORD[0000]

        WORD[0000]              ;LOCATION 0EE
        WORD[0000]

        WORD[0000]              ;LOCATION 0EF
        WORD[0000]

        WORD[0000]              ;LOCATION 0F0
        WORD[0000]

        WORD[0000]              ;LOCATION 0F1
        WORD[0000]

        WORD[0000]              ;LOCATION 0F2
        WORD[0000]

        WORD[0000]              ;LOCATION 0F3
        WORD[0000]

        WORD[0000]              ;LOCATION 0F4
        WORD[0000]

        WORD[0000]              ;LOCATION 0F5
        WORD[0000]

        WORD[0000]              ;LOCATION 0F6
        WORD[0000]

        WORD[0000]              ;LOCATION 0F7
        WORD[0000]

.PAGE
.TOC    "MAIN MEMORY DATA SECTION"
;+++
;       This section lists the data that will be transfered to main memory by
;       the console as part of the initialization performed in test 0.  The
;       TRANSFER DATA routine will point to this data area.  Main memory con-
;       tains the addresses of FPA u-words used for the u-diagnostics, as well
;       as other data used by the tests.
;         Note:  This table is in hexadecimal format i.e. each digit represents
;       four bits, so four digits represnts a full 16 bit word.  The micro-
;       assembler requires that a hexidecimal value that starts with a letter
;       (e.g. FFFF) be preceeded by a 0 (0FFFF).  So some table values will
;       contain 5 hexidecimal digits.  The fifth digit is not actually used.
;
;---

        COUNT.MM[0190]          ;LONGWORD COUNT (NUMBER OF LONGWORDS TO TRANS-
                                ; FER TO MEMORY) DESTINATION IS MAIN MEMORY
                                ; 400 DECIMAL LONGWORDS
        START.ADR[0000]         ;DESTINATION START ADDRESS (START AT MAIN
                                ; MEMORY ADDRESS 0)

        WORD[0FF00]             ;LOCATION 0
        WORD[0FF00]

        WORD[00FF]              ;LOCATION 4
        WORD[00FF]

        WORD[0001]              ;LOCATION 8
        WORD[0000]

        WORD[0002]              ;LOCATION C
        WORD[0000]

        WORD[0004]              ;LOCATION 10
        WORD[0000]

        WORD[0008]              ;LOCATION 14
        WORD[0000]

        WORD[0010]              ;LOCATION 18
        WORD[0000]

        WORD[0020]              ;LOCATION 1C
        WORD[0000]

        WORD[0040]              ;LOCATION 20
        WORD[0000]

        WORD[0080]              ;LOCATION 24
        WORD[0000]

        WORD[0100]              ;LOCATION 28
        WORD[0000]

        WORD[0200]              ;LOCATION 2C
        WORD[0000]

        WORD[0400]              ;LOCATION 30
        WORD[0000]
```

```
        WORD[0800]              ;LOCATION 34
        WORD[0000]

        WORD[1000]              ;LOCATION 38
        WORD[0000]

        WORD[2000]              ;LOCATION 3C
        WORD[0000]

        WORD[4000]              ;LOCATION 40
        WORD[0000]

        WORD[8000]              ;LOCATION 44
        WORD[0000]

        WORD[0000]              ;LOCATION 48
        WORD[0001]

        WORD[0000]              ;LOCATION 4C
        WORD[0002]

        WORD[0000]              ;LOCATION 50
        WORD[0004]

        WORD[0000]              ;LOCATION 54
        WORD[0008]

        WORD[0000]              ;LOCATION 58
        WORD[0010]

        WORD[0000]              ;LOCATION 5C
        WORD[0020]

        WORD[0000]              ;LOCATION 60
        WORD[0040]

        WORD[0000]              ;LOCATION 64
        WORD[0080]

        WORD[0000]              ;LOCATION 68
        WORD[0100]

        WORD[0000]              ;LOCATION 6C
        WORD[0200]

        WORD[0000]              ;LOCATION 70
        WORD[0400]

        WORD[0000]              ;LOCATION 74
        WORD[0800]

        WORD[0000]              ;LOCATION 78
        WORD[1000]

        WORD[0000]              ;LOCATION 7C
        WORD[2000]
        WORD[0000]                 ;LOCATION 80
        WORD[4000]

        WORD[0000]              ;LOCATION 84
        WORD[8000]

        WORD[0000]              ;LOCATION 88
        WORD[0000]

        WORD[0FFFF]             ;LOCATION 8C
        WORD[0FFFF]

        WORD[0101]              ;LOCATION 90
        WORD[0101]

        WORD[0000]              ;LOCATION 94
        WORD[0000]
;
; The following represents the contents of the SPEC DECODE ROM and is used by
; Test 4 in Module 3
;
        WORD[0404]              ;LOCATION 98
        WORD[0404]

        WORD[0404]              ;LOCATION 9C
        WORD[0404]

        WORD[0404]              ;LOCATION 0A0
        WORD[0404]

        WORD[0404]              ;LOCATION 0A4
        WORD[0404]

        WORD[1313]              ;LOCATION 0A8
        WORD[1300]

        WORD[0101]              ;LOCATION 0AC
        WORD[0101]

        WORD[0D0D]              ;LOCATION 0B0
        WORD[0D00]

        WORD[2323]              ;LOCATION 0B4
        WORD[2300]

        WORD[2F2F]              ;LOCATION 0B8
        WORD[2F3A]

        WORD[4343]              ;LOCATION 0BC
        WORD[4351]

        WORD[5B5B]              ;LOCATION 0C0
        WORD[5B64]

        WORD[6D6D]              ;LOCATION 0C4
        WORD[6D79]

        WORD[8585]              ;LOCATION 0C8
```

```
WORD[8592]
```

```
        WORD[9F9F]              ;LOCATION 0CC
        WORD[9FAD]

        WORD[0BBBB]             ;LOCATION 0D0
        WORD[0BBC8]

        WORD[0D5D5]             ;LOCATION 0D4
        WORD[0D5E3]

        WORD[0707]              ;LOCATION 0D8
        WORD[0707]

        WORD[0707]              ;LOCATION 0DC
        WORD[0707]

        WORD[0707]              ;LOCATION 0E0
        WORD[0707]

        WORD[0707]              ;LOCATION 0E4
        WORD[0707]

        WORD[1313]              ;LOCATION 0E8
        WORD[1300]

        WORD[0101]              ;LOCATION 0EC
        WORD[0101]

        WORD[0D0D]              ;LOCATION 0F0
        WORD[0D00]

        WORD[2323]              ;LOCATION 0F4
        WORD[2300]

        WORD[2F2F]              ;LOCATION 0F8
        WORD[2F3A]

        WORD[4343]              ;LOCATION 0FC
        WORD[4351]

        WORD[5B5B]              ;LOCATION 100
        WORD[5B64]

        WORD[6D6D]              ;LOCATION 104
        WORD[6D79]

        WORD[8585]              ;LOCATION 108
        WORD[8592]

        WORD[9F9F]              ;LOCATION 10C
        WORD[9FAD]

        WORD[0BBBB]             ;LOCATION 110
        WORD[0BBC8]

        WORD[0D5D5]             ;LOCATION 114
        WORD[0D5E3]
```

```
        WORD[0000]                      ;LOCATION 118
        WORD[0000]

        WORD[0000]                      ;LOCATION 11C
        WORD[0000]

        WORD[0000]                      ;LOCATION 120
        WORD[0000]

        WORD[0000]                      ;LOCATION 124
        WORD[0000]

        WORD[1313]                      ;LOCATION 128
        WORD[1300]

        WORD[0000]                      ;LOCATION 12C
        WORD[0000]

        WORD[0D0D]                      ;LOCATION 130
        WORD[0D00]

        WORD[2323]                      ;LOCATION 134
        WORD[2300]

        WORD[2F2F]                      ;LOCATION 138
        WORD[2F3A]

        WORD[4343]                      ;LOCATION 13C
        WORD[4351]

        WORD[5B5B]                      ;LOCATION 140
        WORD[5B64]

        WORD[6D6D]                      ;LOCATION 144
        WORD[6D79]

        WORD[8585]                      ;LOCATION 148
        WORD[8592]

        WORD[9F9F]                      ;LOCATION 14C
        WORD[9FAD]

        WORD[0BBBB]                     ;LOCATION 150
        WORD[0BBC8]

        WORD[0D5D5]                     ;LOCATION 154
        WORD[0D5E3]

        WORD[0FFFF]                     ;LOCATION 158
        WORD[0FFFF]

        WORD[0FFFF]                     ;LOCATION 15C
        WORD[0FFFF]

        WORD[0FFFF]                     ;LOCATION 160
        WORD[0FFFF]
        WORD[0FFFF]             ;LOCATION 164
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 168
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 16C
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 170
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 174
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 178
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 17C
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 180
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 184
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 188
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 18C
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 190
        WORD[0FFFF]

        WORD[0FFFF]             ;LOCATION 194
        WORD[0FFFF]

        WORD[0000]              ;LOCATION 198
        WORD[0000]

        WORD[0000]              ;LOCATION 19C
        WORD[0000]

        WORD[0000]              ;LOCATION 1A0
        WORD[0000]

        WORD[0000]              ;LOCATION 1A4
        WORD[0000]

        WORD[1313]              ;LOCATION 1A8
        WORD[1300]

        WORD[0202]              ;LOCATION 1AC
        WORD[0202]

        WORD[0D0D]              ;LOCATION 1B0
```

```
WORD[0D00]
```

```
        WORD[2323]              ;LOCATION 1B4
        WORD[2300]

        WORD[2F2F]              ;LOCATION 1B8
        WORD[2F3A]

        WORD[4343]              ;LOCATION 1BC
        WORD[4351]

        WORD[5B5B]              ;LOCATION 1C0
        WORD[5B64]

        WORD[6D6D]              ;LOCATION 1C4
        WORD[6D79]

        WORD[8585]              ;LOCATION 1C8
        WORD[8592]

        WORD[9F9F]              ;LOCATION 1CC
        WORD[9FAD]

        WORD[0BBBB]             ;LOCATION 1D0
        WORD[0BBC8]

        WORD[0D5D5]             ;LOCATION 1D4
        WORD[0D5E3]

        WORD[0000]              ;LOCATION 1D8
        WORD[0000]

        WORD[0000]              ;LOCATION 1DC
        WORD[0000]

        WORD[0000]              ;LOCATION 1E0
        WORD[0000]

        WORD[0000]              ;LOCATION 1E4
        WORD[0000]

        WORD[2222]              ;LOCATION 1E8
        WORD[2222]

        WORD[0000]              ;LOCATION 1EC
        WORD[0000]

        WORD[0D0D]              ;LOCATION 1F0
        WORD[0D00]

        WORD[2323]              ;LOCATION 1F4
        WORD[2300]

        WORD[2F2F]              ;LOCATION 1F8
        WORD[2F3A]

        WORD[4343]              ;LOCATION 1FC
        WORD[4351]
```

```
        WORD[5B5B]              ;LOCATION 200
        WORD[5B64]

        WORD[6D6D]              ;LOCATION 204
        WORD[6D79]

        WORD[8585]              ;LOCATION 208
        WORD[8592]

        WORD[9F9F]              ;LOCATION 20C
        WORD[9FAD]

        WORD[0BBBB]             ;LOCATION 210
        WORD[0BBC8]

        WORD[0D5D5]             ;LOCATION 214
        WORD[0D5E3]

        WORD[2B2B]              ;LOCATION 218
        WORD[2C2C]

        WORD[3131]              ;LOCATION 21C
        WORD[3838]

        WORD[4141]              ;LOCATION 220
        WORD[4949]

        WORD[5353]              ;LOCATION 224
        WORD[5C5C]

        WORD[2B2B]              ;LOCATION 228
        WORD[2C2C]

        WORD[3131]              ;LOCATION 22C
        WORD[3838]

        WORD[4141]              ;LOCATION 230
        WORD[4949]

        WORD[5353]              ;LOCATION 234
        WORD[5C5C]

        WORD[2B2B]              ;LOCATION 238
        WORD[2C2C]

        WORD[3131]              ;LOCATION 23C
        WORD[3838]

        WORD[4141]              ;LOCATION 240
        WORD[4949]

        WORD[5353]              ;LOCATION 244
        WORD[5C5C]

        WORD[2B2B]              ;LOCATION 248
        WORD[2C2C]
        WORD[3131]                ;LOCATION 24C
        WORD[3838]

        WORD[4141]               ;LOCATION 250
        WORD[4949]

        WORD[5353]               ;LOCATION 254
        WORD[5C5C]

        WORD[0000]               ;LOCATION 258
        WORD[0000]

        WORD[0000]               ;LOCATION 25C
        WORD[0000]

        WORD[0000]               ;LOCATION 260
        WORD[0000]

        WORD[0000]               ;LOCATION 264
        WORD[0000]

        WORD[0101]               ;LOCATION 268
        WORD[0101]

        WORD[0101]               ;LOCATION 26C
        WORD[0101]

        WORD[0101]               ;LOCATION 270
        WORD[0101]

        WORD[0101]               ;LOCATION 274
        WORD[0101]

        WORD[0202]               ;LOCATION 278
        WORD[0202]

        WORD[0202]               ;LOCATION 27C
        WORD[0202]

        WORD[0202]               ;LOCATION 280
        WORD[0202]

        WORD[0202]               ;LOCATION 284
        WORD[0202]

        WORD[0303]               ;LOCATION 288
        WORD[0303]

        WORD[0303]               ;LOCATION 28C
        WORD[0303]

        WORD[0303]               ;LOCATION 290
        WORD[0303]

        WORD[0303]               ;LOCATION 294
        WORD[0303]
;
; The following represents the contents of the OPCODE DECODE ROM and is used by
; Test 5 of Module 3
```

```
;
        WORD[0402]                      ;LOCATION 298
        WORD[1501]

        WORD[0501]                      ;LOCATION 29C
        WORD[0601]

        WORD[0401]                      ;LOCATION 2A0
        WORD[1501]

        WORD[0439]                      ;LOCATION 2A4
        WORD[0E40]

        WORD[2708]                      ;LOCATION 2A8
        WORD[0418]

        WORD[0720]                      ;LOCATION 2AC
        WORD[3040]

        WORD[8D01]                      ;LOCATION 2B0
        WORD[0B101]

        WORD[0B206]                     ;LOCATION 2B4
        WORD[9802]

        WORD[7D01]                      ;LOCATION 2B8
        WORD[8001]

        WORD[8301]                      ;LOCATION 2BC
        WORD[8601]

        WORD[0BD02]                     ;LOCATION 2C0
        WORD[0BE10]

        WORD[0CC10]                     ;LOCATION 2C4
        WORD[0C910]

        WORD[0DB10]                     ;LOCATION 2C8
        WORD[0CF10]

        WORD[0D310]                     ;LOCATION 2CC
        WORD[8E02]

        WORD[9002]                      ;LOCATION 2D0
        WORD[9202]

        WORD[9402]                      ;LOCATION 2D4
        WORD[9602]

        WORD[0BD04]                     ;LOCATION 2D8
        WORD[0A802]

        WORD[0B208]                     ;LOCATION 2DC
        WORD[0BB01]

        WORD[0BC01]                     ;LOCATION 2E0
        WORD[7D01]
```

```
        WORD[8001]                      ;LOCATION 2E4
        WORD[8301]

        WORD[8901]                      ;LOCATION 2E8
        WORD[0BD02]

        WORD[0C210]                     ;LOCATION 2EC
        WORD[0CC10]

        WORD[0C910]                     ;LOCATION 2F0
        WORD[0DB10]

        WORD[0CF10]                     ;LOCATION 2F4
        WORD[0D710]

        WORD[0BD10]                     ;LOCATION 2F8
        WORD[0EAD8]

        WORD[0D900]                     ;LOCATION 2FC
        WORD[0000]

        WORD[0DCDB]                     ;LOCATION 300
        WORD[1211]

        WORD[0E666]                     ;LOCATION 304
        WORD[0DAD9]

        WORD[3818]                      ;LOCATION 308
        WORD[0000]

        WORD[0000]                      ;LOCATION 30C
        WORD[0000]

        WORD[0000]                      ;LOCATION 310
        WORD[0000]

        WORD[0000]                      ;LOCATION 314
        WORD[0000]

        WORD[0000]                      ;LOCATION 318
        WORD[0807]

        WORD[0A09]                      ;LOCATION 31C
        WORD[0B0F]

        WORD[0C10]                      ;LOCATION 320
        WORD[211D]

        WORD[2625]                      ;LOCATION 324
        WORD[221E]

        WORD[201F]                      ;LOCATION 328
        WORD[0000]

        WORD[0A096]                     ;LOCATION 32C
        WORD[0504]
        WORD[060E]                  ;LOCATION 330
```

```
	WORD[27E7]

	WORD[2423]		;LOCATION 334
	WORD[0E382]

	WORD[0402]		;LOCATION 338
	WORD[0C03]

	WORD[0900]		;LOCATION 33C
	WORD[4E49]

	WORD[2D28]		;LOCATION 340
	WORD[0ADA8]

	WORD[0B6B2]		;LOCATION 344
	WORD[0FCFC]

	WORD[3FFC]		;LOCATION 348
	WORD[6805]

	WORD[2904]		;LOCATION 34C
	WORD[0DDF1]

	WORD[0DFCB]		;LOCATION 350
	WORD[0DFDA]

	WORD[0DFDF]		;LOCATION 354
	WORD[0A064]

	WORD[0E9E8]		;LOCATION 358
	WORD[2113]

	WORD[1E10]		;LOCATION 35C
	WORD[5F52]

	WORD[3F31]		;LOCATION 360
	WORD[908B]

	WORD[9995]		;LOCATION 364
	WORD[0F3F3]

	WORD[77F3]		;LOCATION 368
	WORD[7611]

	WORD[1D10]		;LOCATION 36C
	WORD[0DEF2]

	WORD[0DFEB]		;LOCATION 370
	WORD[0E001]

	WORD[0FDE6]		;LOCATION 374
	WORD[0000]

	WORD[0E1E0]		;LOCATION 378
	WORD[002A]

	WORD[003D]		;LOCATION 37C
```

```
WORD[1A00]
```

```
        WORD[762C]                      ;LOCATION 380
        WORD[4B43]

        WORD[544C]                      ;LOCATION 384
        WORD[5E56]

        WORD[4C6A]                      ;LOCATION 388
        WORD[0000]

        WORD[0CD7A]                     ;LOCATION 38C
        WORD[0C600]

        WORD[0000]                      ;LOCATION 390
        WORD[8D86]

        WORD[7D7E]                      ;LOCATION 394
        WORD[0E3D1]

        WORD[0401]                      ;LOCATION 398
        WORD[002A]

        WORD[003D]                      ;LOCATION 39C
        WORD[200C]

        WORD[7D37]                      ;LOCATION 3A0
        WORD[4B43]

        WORD[544C]                      ;LOCATION 3A4
        WORD[5E56]

        WORD[4C6A]                      ;LOCATION 3A8
        WORD[0000]

        WORD[0CD7A]                     ;LOCATION 3AC
        WORD[0C600]

        WORD[0000]                      ;LOCATION 3B0
        WORD[3C3D]

        WORD[0A081]                     ;LOCATION 3B4
        WORD[0DEDB]

        WORD[0DDDC]                     ;LOCATION 3B8
        WORD[002A]

        WORD[003D]                      ;LOCATION 3BC
        WORD[2815]

        WORD[8B47]                      ;LOCATION 3C0
        WORD[4B43]

        WORD[544C]                      ;LOCATION 3C4
        WORD[5E56]

        WORD[4C6A]                      ;LOCATION 3C8
        WORD[0000]
```

```
        WORD[0CD7A]             ;LOCATION 3CC
        WORD[0C600]

        WORD[0000]              ;LOCATION 3D0
        WORD[0C8C1]

        WORD[0DFE4]             ;LOCATION 3D4
        WORD[5F00]

        WORD[0400]              ;LOCATION 3D8
        WORD[6964]

        WORD[7E6E]              ;LOCATION 3DC
        WORD[8676]

        WORD[908E]              ;LOCATION 3E0
        WORD[0502]

        WORD[2221]              ;LOCATION 3E4
        WORD[1314]

        WORD[0809]              ;LOCATION 3E8
        WORD[0E3E5]

        WORD[0000]              ;LOCATION 3EC
        WORD[0000]

        WORD[0B1A3]             ;LOCATION 3F0
        WORD[0D13]

        WORD[0000]              ;LOCATION 3F4
        WORD[0E3E2]

        WORD[0E5E4]             ;LOCATION 3F8
        WORD[0BCBC]

        WORD[0BCBD]             ;LOCATION 3FC
        WORD[0D8BE]

        WORD[0BCBC]             ;LOCATION 400
        WORD[7979]

        WORD[0BC66]             ;LOCATION 404
        WORD[8989]

        WORD[8686]              ;LOCATION 408
        WORD[0B0C1]

        WORD[0AFAF]             ;LOCATION 40C
        WORD[0AFAF]

        WORD[0E6ED]             ;LOCATION 410
        WORD[0AFAF]

        WORD[0AFAF]             ;LOCATION 414
        WORD[0AFAF]
```

```
WORD[0AFAF]             ;LOCATION 418
```

```
        WORD[7979]

        WORD[7979]               ;LOCATION 41C
        WORD[7979]

        WORD[7979]               ;LOCATION 420
        WORD[8C8C]

        WORD[8C8C]               ;LOCATION 424
        WORD[8C8C]

        WORD[8C8C]               ;LOCATION 428
        WORD[0B5C2]

        WORD[7373]               ;LOCATION 42C
        WORD[7979]

        WORD[7979]               ;LOCATION 430
        WORD[8CBC]

        WORD[8C8C]               ;LOCATION 434
        WORD[435C]

        WORD[8686]               ;LOCATION 438
        WORD[1D0B]

        WORD[1D0B]               ;LOCATION 43C
        WORD[1D0B]

        WORD[1D0B]               ;LOCATION 440
        WORD[4C4C]

        WORD[4C4C]               ;LOCATION 444
        WORD[8D8D]

        WORD[598D]               ;LOCATION 448
        WORD[4C59]

        WORD[5959]               ;LOCATION 44C
        WORD[0BC4C]

        WORD[0BC4C]              ;LOCATION 450
        WORD[0BCBC]

        WORD[0BCBC]              ;LOCATION 454
        WORD[8686]

        WORD[0BCBC]              ;LOCATION 458
        WORD[3C2B]

        WORD[3C2B]               ;LOCATION 45C
        WORD[3C2B]

        WORD[3C2B]               ;LOCATION 460
        WORD[6262]

        WORD[6262]               ;LOCATION 464
```

```
WORD[8D8D]
```

```
        WORD[718D]                      ;LOCATION 468
        WORD[6271]

        WORD[7171]                      ;LOCATION 46C
        WORD[0BC62]

        WORD[0BC62]                     ;LOCATION 470
        WORD[8966]

        WORD[8989]                      ;LOCATION 474
        WORD[9700]

        WORD[0BCBC]                     ;LOCATION 478
        WORD[0C404]

        WORD[0CE04]                     ;LOCATION 47C
        WORD[4324]

        WORD[4324]                      ;LOCATION 480
        WORD[3304]

        WORD[3304]                      ;LOCATION 484
        WORD[3304]

        WORD[665C]                      ;LOCATION 488
        WORD[9891]

        WORD[335C]                      ;LOCATION 48C
        WORD[5CAA]

        WORD[0A5A0]                     ;LOCATION 490
        WORD[6B6B]

        WORD[5C5C]                      ;LOCATION 494
        WORD[435C]

        WORD[8686]                      ;LOCATION 498
        WORD[0C404]

        WORD[0CE04]                     ;LOCATION 49C
        WORD[4324]

        WORD[4324]                      ;LOCATION 4A0
        WORD[3304]

        WORD[3304]                      ;LOCATION 4A4
        WORD[3304]

        WORD[665C]                      ;LOCATION 4A8
        WORD[9891]

        WORD[335C]                      ;LOCATION 4AC
        WORD[5CAA]

        WORD[0A5A0]                     ;LOCATION 4B0
        WORD[6666]
```

```
        WORD[6666]              ;LOCATION 4B4
        WORD[8989]

        WORD[8989]              ;LOCATION 4B8
        WORD[0C404]

        WORD[0CE04]             ;LOCATION 4BC
        WORD[5014]

        WORD[5014]              ;LOCATION 4C0
        WORD[3304]

        WORD[3304]              ;LOCATION 4C4
        WORD[3304]

        WORD[665C]              ;LOCATION 4C8
        WORD[9891]

        WORD[335C]              ;LOCATION 4CC
        WORD[5CAA]

        WORD[0A5A0]             ;LOCATION 4D0
        WORD[0404]

        WORD[8943]              ;LOCATION 4D4
        WORD[5CC3]

        WORD[86DF]              ;LOCATION 4D8
        WORD[7A7A]

        WORD[7A7A]              ;LOCATION 4DC
        WORD[7A7A]

        WORD[7A7A]              ;LOCATION 4E0
        WORD[6666]

        WORD[6666]              ;LOCATION 4E4
        WORD[6666]

        WORD[6666]              ;LOCATION 4E8
        WORD[4343]

        WORD[0F9FA]             ;LOCATION 4EC
        WORD[0C0BF]

        WORD[8080]              ;LOCATION 4F0
        WORD[7979]

        WORD[6686]              ;LOCATION 4F4
        WORD[0BCBC]

        WORD[0BCBC]             ;LOCATION 4F8
        WORD[0000]

        WORD[0000]              ;LOCATION 4FC
        WORD[0000]
        WORD[0308]              ;LOCATION 500
        WORD[0702]

        WORD[0C01]                      ;LOCATION 504
        WORD[0303]

        WORD[0202]                      ;LOCATION 508
        WORD[0004]

        WORD[0402]                      ;LOCATION 50C
        WORD[0002]

        WORD[0602]                      ;LOCATION 510
        WORD[0102]

        WORD[0202]                      ;LOCATION 514
        WORD[0302]

        WORD[0404]                      ;LOCATION 518
        WORD[0701]

        WORD[0401]                      ;LOCATION 51C
        WORD[0701]

        WORD[0401]                      ;LOCATION 520
        WORD[0701]

        WORD[0601]                      ;LOCATION 524
        WORD[0702]

        WORD[0604]                      ;LOCATION 528
        WORD[0002]

        WORD[0402]                      ;LOCATION 52C
        WORD[0701]

        WORD[0401]                      ;LOCATION 530
        WORD[0701]

        WORD[0402]                      ;LOCATION 534
        WORD[0701]

        WORD[0302]                      ;LOCATION 538
        WORD[0704]

        WORD[0C0C]                      ;LOCATION 53C
        WORD[0001]

        WORD[0401]                      ;LOCATION 540
        WORD[0C01]

        WORD[0F02]                      ;LOCATION 544
        WORD[0C06]

        WORD[0F01]                      ;LOCATION 548
        WORD[0401]

        WORD[0F03]                      ;LOCATION 54C
        WORD[0004]
        WORD[0C0C]              ;LOCATION 550
        WORD[0001]

        WORD[0401]              ;LOCATION 554
        WORD[0C01]

        WORD[0F02]              ;LOCATION 558
        WORD[0C06]

        WORD[0F01]              ;LOCATION 55C
        WORD[0002]

        WORD[0C02]              ;LOCATION 560
        WORD[0F04]

        WORD[0002]              ;LOCATION 564
        WORD[0102]

        WORD[0004]              ;LOCATION 568
        WORD[0306]

        WORD[0001]              ;LOCATION 56C
        WORD[0201]

        WORD[0301]              ;LOCATION 570
        WORD[0201]

        WORD[0303]              ;LOCATION 574
        WORD[0002]

        WORD[0101]              ;LOCATION 578
        WORD[0002]

        WORD[0306]              ;LOCATION 57C
        WORD[0402]

        WORD[0502]              ;LOCATION 580
        WORD[0404]

        WORD[0706]              ;LOCATION 584
        WORD[0401]

        WORD[0601]              ;LOCATION 588
        WORD[0701]

        WORD[0601]              ;LOCATION 58C
        WORD[0703]

        WORD[0402]              ;LOCATION 590
        WORD[0501]

        WORD[0402]              ;LOCATION 594
        WORD[0706]

        WORD[0C02]              ;LOCATION 598
        WORD[0D02]
```

```
        WORD[0C04]              ;LOCATION 59C
        WORD[0F06]

        WORD[0C01]              ;LOCATION 5A0
        WORD[0E01]

        WORD[0F01]              ;LOCATION 5A4
        WORD[0E01]

        WORD[0F03]              ;LOCATION 5A8
        WORD[0C02]

        WORD[0D01]              ;LOCATION 5AC
        WORD[0C01]

        WORD[0D01]              ;LOCATION 5B0
        WORD[0F10]

        WORD[0C02]              ;LOCATION 5B4
        WORD[0E02]

        WORD[0F02]              ;LOCATION 5B8
        WORD[0C01]

        WORD[0F05]              ;LOCATION 5BC
        WORD[0C02]

        WORD[0001]              ;LOCATION 5C0
        WORD[0F01]

        WORD[0301]              ;LOCATION 5C4
        WORD[0F05]

        WORD[0701]              ;LOCATION 5C8
        WORD[0003]

        WORD[0702]              ;LOCATION 5CC
        WORD[0502]

        WORD[0703]              ;LOCATION 5D0
        WORD[0501]

        WORD[0401]              ;LOCATION 5D4
        WORD[0713]

        WORD[0510]              ;LOCATION 5D8
        WORD[0730]

        WORD[0410]              ;LOCATION 5DC
        WORD[0710]

        WORD[0602]              ;LOCATION 5E0
        WORD[0102]

        WORD[0202]              ;LOCATION 5E4
        WORD[0302]
        WORD[0702]              ;LOCATION 5E8
        WORD[0301]

        WORD[0101]                ;LOCATION 5EC
        WORD[0001]

        WORD[0703]                ;LOCATION 5F0
        WORD[0310]

        WORD[0110]                ;LOCATION 5F4
        WORD[0330]

        WORD[0510]                ;LOCATION 5F8
        WORD[0710]

        WORD[0000]                ;LOCATION 5FC
        WORD[0000]

        WORD[0000]                ;LOCATION 600
        WORD[0000]

        WORD[0000]                ;LOCATION 604
        WORD[0000]

        WORD[0000]                ;LOCATION 608
        WORD[0000]

        WORD[0000]                ;LOCATION 60C
        WORD[0000]

        WORD[0000]                ;LOCATION 610
        WORD[0000]

        WORD[0000]                ;LOCATION 614
        WORD[0000]

        WORD[0000]                ;LOCATION 618
        WORD[0000]

        WORD[0000]                ;LOCATION 61C
        WORD[0000]

        WORD[0000]                ;LOCATION 620
        WORD[0000]

        WORD[0000]                ;LOCATION 624
        WORD[0000]

        WORD[0000]                ;LOCATION 628
        WORD[0000]

        WORD[0000]                ;LOCATION 62C
        WORD[0000]

        WORD[0000]                ;LOCATION 630
        WORD[0000]

        WORD[0000]                ;LOCATION 634
```

```
    WORD[0000]
```

```
        WORD[0000]                      ;LOCATION 638
        WORD[0000]

        WORD[0000]                      ;LOCATION 63C
        WORD[0000]

        WORD[0000]                      ;LOCATION 640
        WORD[0000]

        WORD[0000]                      ;LOCATION 644
        WORD[0000]

        WORD[0000]                      ;LOCATION 648
        WORD[0000]

        WORD[0000]                      ;LOCATION 64C
        WORD[0000]

        WORD[0000]                      ;LOCATION 650
        WORD[0000]

        WORD[0000]                      ;LOCATION 654
        WORD[0000]

        WORD[0000]                      ;LOCATION 658
        WORD[0000]

        WORD[0000]                      ;LOCATION 65C
        WORD[0000]

        WORD[0000]                      ;LOCATION 660
        WORD[0000]

        WORD[0000]                      ;LOCATION 664
        WORD[0000]

        WORD[0000]                      ;LOCATION 668
        WORD[0000]

        WORD[0000]                      ;LOCATION 66C
        WORD[0000]

        WORD[0000]                      ;LOCATION 670
        WORD[0000]

        WORD[0000]                      ;LOCATION 674
        WORD[0000]

        WORD[0000]                      ;LOCATION 678
        WORD[0000]

        WORD[0000]                      ;LOCATION 67C
        WORD[0000]

        WORD[0000]                      ;LOCATION 680
        WORD[0000]
```

```
        WORD[0000]                     ;LOCATION 684
        WORD[0000]

        WORD[0000]                     ;LOCATION 688
        WORD[0000]

        WORD[0000]                     ;LOCATION 68C
        WORD[0000]

        WORD[0000]                     ;LOCATION 690
        WORD[0000]

        WORD[0000]                     ;LOCATION 694
        WORD[0000]

        WORD[0000]                     ;LOCATION 698
        WORD[0000]

        WORD[0000]                     ;LOCATION 69C
        WORD[0000]

        WORD[0000]                     ;LOCATION 6A0
        WORD[0000]

        WORD[0000]                     ;LOCATION 6A4
        WORD[0000]

        WORD[0000]                     ;LOCATION 6A8
        WORD[0000]

        WORD[0000]                     ;LOCATION 6AC
        WORD[0000]

        WORD[0000]                     ;LOCATION 6B0
        WORD[0000]

        WORD[0000]                     ;LOCATION 6B4
        WORD[0000]

        WORD[0000]                     ;LOCATION 6B8
        WORD[0000]

        WORD[0000]                     ;LOCATION 6BC
        WORD[0000]

        WORD[0000]                     ;LOCATION 6C0
        WORD[0000]

        WORD[0000]                     ;LOCATION 6C4
        WORD[0000]

        WORD[0000]                     ;LOCATION 6C8
        WORD[0000]

        WORD[0000]                     ;LOCATION 6CC
        WORD[0000]
        WORD[0000]                ;LOCATION 6D0
        WORD[0000]

        WORD[0000]                 ;LOCATION 6D4
        WORD[0000]

        WORD[0000]                 ;LOCATION 6D8
        WORD[0000]

        WORD[0000]                 ;LOCATION 6DC
        WORD[0000]

        WORD[0000]                 ;LOCATION 6E0
        WORD[0000]

        WORD[0000]                 ;LOCATION 6E4
        WORD[0000]

        WORD[0000]                 ;LOCATION 6E8
        WORD[0000]

        WORD[0000]                 ;LOCATION 6EC
        WORD[0000]

        WORD[0000]                 ;LOCATION 6F0
        WORD[0000]

        WORD[0000]                 ;LOCATION 6F4
        WORD[0000]

        WORD[0000]                 ;LOCATION 6F8
        WORD[0000]

        WORD[0000]                 ;LOCATION 6FC
        WORD[0000]
.REGION/600,6FF
.PAGE   "WCS SUBROUTINES FOR CONSOLE USE"
.TOC    "  GENERATE TRANSFER DATA"
;
;       This routine is used by the diagnostic monitor to load WR 0 with a
;       data pattern from the console.  The monitor will set CONSOLE ATTN if
;       a 1 is to be generated, or clear CONSOLE ATTN if a 0 is to be gen-
;       erated.  It will then single step this routine to shift the data bit
;       into WR 0.  This routine loads a 32 bit value much more quickly than
;       shifting each instruction into the CSR from the monitor, and is used
;       mainly to set up data to load in LS for constants.
;
;
;                                  ***WARNING***
;
;       This routine must start at 600 and end at 605.  DO NOT move this
;       routine to any other area!!
;
;

GEN.XFER.DATA:
        CLR WR[0]                            ;CLEAR A WORKING REG
```

```
	COM WR[0]			;NOW WR 0 IS ALL ONES
```

```
LOOP.XFER:
        SKIP.IF[CONSOLE.ATTN]              ;SKIP NEXT INSTRUCTION IS CONSOLE ATTN
                                           ; IS SET (GENERATING A 1)
        ASHL WR[0],                        ;SHIFT A 0 INTO BIT 0 OF WR 0
         SKIP                              ; AND SKIP ROL
        ROL WR[0]                          ;SHIFT A 1 INTO BIT 0 OF WR 0
        JMP [LOOP.XFER]                    ;REPEAT


;
;       DIAGNOSTIC VERSION NUMBER IS STORED HERE
;

;                       ***WARNING***
;
;       These words must be at location 606 and 607.  DO NOT move this word
;       to any other area!!
;
;


        WORD[0110]                         ;VERSION 01.10
        ASCII [CD]                         ;LAST 2 LETTERS OF FILE NAME [ENKCD]

.PAGE   "  DEPOSIT MCT CSR ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       The deposit to CSR routine is used to write the memory controller's
;       control and status register.  The memory controller has three CSRs
;       named CSR 0, CSR 1, and CSR 2.
;         CSR 0 contains checkbits or syndrome bits returned from the ECC logic
;       after certain diagnostic routines.  It is NOT writable directly with
;       this routine.
;         CSR 1 contains error bits set if an error occurs on a memory access.
;       It does contain five maintenance bits (bits 29-25) which are writable.
;       There are also seven checkbits (bits 6-0) which are writable, but not
;       readable.  These bits are used to write checkbits into the ECC chips.
;         CSR 2 contains error bits set if a UNIBUS error occurs on a UNIBUS
;       access.  These are read only bits and are NOT writable by this routine.
;         Therefore, CSR 1 is the only writable CSR, and only bits 29-25 can be
;       both written and read back.  This routine will thus force CSR 1 on a
;       deposit to CSR.
;         This routine will write LS 5 with data to access CSR 1, then write
;       the data residing in LS 6 into the CSR.  It will then issue CPU ATTN
;       to inform the console that it has completed the function.
;         NOTE:  The error bits of CSR 1 are cleared on any write to CSR 1.
;       These are bits 31, 30 and 23-14.Bits 13-0 and bit 24 are unused.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 50) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS location 6 must have been written by the console with the desired
;       data to be deposited in CSR 1 previous to executing this routine.
```

```
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       CSR 1 bits 29-25 and 6-0 are written with data from LS 6.
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       WR 0 will be modified by this routine.
;       LS 5 will be modified by this routine.
;       CSR 1 bits 31, 30 and 23-14 are cleared.
;
;---

DEPOSIT.CSR:

        MOV LS[CSR1] TO WR[0]           ;SET BIT 2 IN WR 0 AS CSR NUMBER TO
                                        ; WRITE (BITS 2 AND 3 ARE CSR NUMBER)
        MOV WR[0] TO LS[T5.SUB]         ;PUT CSR NUMBER IN LS LOCATION 5 (CSR
                                        ; 1)
        MEM.REQ[WRITE.CSR] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                                        ; ACCESS CSR 1
        WRITE.MEM LS[T6.SUB]            ;SEND DATA IN LS LOCATION 6 TO CSR 1
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
.PAGE   "  EXAMINE MCT CSR ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       The examine CSR routine reads the memory controller's control and
;       status register.  There are 3 CSRs labeled CSR 0, CSR 1 and CSR 2.
;         CSR 0 contains checkbits or syndrome bits from the ECC logic.
;       These are contained in bits 6-0 of the CSR.  Other bits are UNPRE-
;       DICTABLE.  Bits 6-0 are only written by special diagnostic routines
;       (they are not writtable by the DEPOSIT CSR command).
;         CSR 1 contains error bits and maintenance bits.  The error bits are
;       31, 30, and 23-14.  Maintenance bits are 29-25.  The other bits are
;       UNPREDICTABLE.  The error bits are written during memory functions
;       only and are not writable via the DEPOSIT CSR command. The maintenance
;       bits are writable.  Note that a DEPOSIT CSR command will write the
;       maintenance bits and clear all error bits.
;         CSR 2 contains three bits (bits 16-14) which are readable.  These are
;       set when a UNIBUS error occurs on a UNIBUS access.  They are not
;       directly writable with the DEPOSIT CSR command.
;         The console loads LS 5 with the CSR number (0, 1, or 2) to be read.
;       This routine rotates LS 5 two places left to position it correctly.
;       It then executes a read CSR and places the data in LS 6 for the console
;       to read.
;
; CALLING SEQUENCE:
```

```
;
```

```
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 51) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS 5 contains the CSR number to be read.
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       LS 6 - Data read from CSR selected.
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       LS 5 is rotated 2 places left.
;
;---

EXAMINE.CSR:

        MOV LS[T5.SUB] TO WR[0]         ;GET CSR NUMBER FROM LS 5
        SHL2 WR[0]                      ;SHIFT NUMBER 2 PLACES LEFT TO GET
                                        ; CSR NUMBER IN BITS 2 AND 3
        MOV WR[0] TO LS[T5.SUB]         ;PUT SHIFTED NUMBER IN LS 5

        MEM.REQ[READ.CSR] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                                        ; ACCESS CSR SPECIFIED
        MOV MEM.DATA TO WR[0]           ;READ CSR SELECTED

CHECK.CSR2:
        MOV LS[T5.SUB] TO WR[1]         ;GET SHIFTED CSR NUMBER
        CMP LS[#8] WITH WR[1],          ;SEE IF CSR 2 WAS SELECTED
         DT(LONG)&SET.ALU.CC            ; SET UP CONDITION CODES
        JMP.IF[NEQ] TO [CHECK.CSR1]     ;JUMP IF CSR 2 WAS NOT SELECTED
        BIC LS[CSR2.MASK] TO WR[0]      ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
                                        ; CSR 2.
CHECK.CSR1:
        CMP LS[#4] WITH WR[1],          ;SEE IF CSR 1 WAS SELECTED
         DT(LONG)&SET.ALU.CC            ; SET UP CONDITION CODES
        JMP.IF[NEQ] TO [CHECK.CSR0]     ;JUMP IF CSR 1 WAS NOT SELECTED
        BIC LS[CSR1.MASK] TO WR[0]      ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
                                        ; CSR 1.
CHECK.CSR0:
        CMP LS[#0] WITH WR[1],          ;SEE IF CSR 0 WAS SELECTED
         DT(LONG)&SET.ALU.CC            ; SET UP CONDITION CODES
        JMP.IF[NEQ] TO [LOAD.LS6]       ;JUMP IF CSR 0 WAS NOT SELECTED
        BIC LS[CSR0.MASK] TO WR[0]      ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
                                        ; CSR 0.
        BIC LS[BIT7] TO WR[0]           ;BIT 7 IS ALSO UNUSED
```

```
LOAD.LS6:
        MOV WR[0] TO LS[T6.SUB]          ;LOAD CSR DATA INTO LS 6 FOR PRINTOUT
        JMP [WAIT.SUB]                   ;ISSUE CPU ATTN AND WAIT FOR RESPONSE
.page   "  DEPOSIT MCT TRANSLATION BUFFER ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       This routine allows a write to the translation buffer portion of the
;       memory controller.  The buffer area contains 128 decimal locations
;       addressed from 0-7F(H).  The remainder of the TB RAM is either unused
;       or contains the UNIBUS MAP.  The DEPOSIT UB command is used to write
;       the UNIBUS map portion of the TB RAMs.  The address specified with the
;       DEPOSIT command will be the actual RAM address accessed.  This routine
;       will do any shifting necessary on the address specified to position it
;       correctly.  The address specified has been loaded by the console in LS
;       location 5 prior to executing this routine.  The address must be in HEX
;       format.
;         The data specified has been placed in LS 6 by the console prior to
;       executing this routine.  Bits 07-01 will be the Valid, Prot, Modify,
;       and Byte Offset bits (there are four Prot bits).  Bits 22-08 will
;       contain the PA bits from PA 23 through PA 09 respectively.  Bits 31
;       through 23 and bit 0 are unused. This routine will do any shifting nec-
;       essary to write the bits in TB as described.  The data specified must
;       be in HEX format.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 52) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS 5 - TB address to be written (range 0-7F)
;       LS 6 - TB data to write (22 bits, from 1-22).  Bits 0 and 23-31 are
;              ignored.
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       TB will be written with specified data at specified address
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       WR 0 will be modified
;       WR 1 will be modified
;
;---

DEPOSIT.TB:
        JSR [SHIFT.TB.ADR]               ;GO TO SUBROUTINE TO POSITION TB AD-
```

```
                        ; DRESS CORRECTLY
```

```
        MOV LS[T6.SUB] TO WR[0]         ;GET DATA FROM LS 6
        BIC LS[#FF] TO WR[0]            ;CLEAR LOW BYTE OF DATA SPECIFIED.
                                        ; LOW BYTE WILL BE PUT IN BITS 24-31
                                        ; LATER
        JSR [SHIFT.LOOP]                ;GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
                                        ; PLACES RIGHT
        SWAP LS[T6.SUB] WITH WR[0]      ;SWAP ORIGINAL DATA SPECIFIED WITH MOD-
                                        ; IFIED DATA.  NOW LS 6 CONTAINS BITS
                                        ; 8-22 IN BITS 0-14 AS REQUIRED.  WR 0
                                        ; CONTAINS ORIGINAL DATA SPECIFIED
        BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE OF ORIGINAL
                                        ; DATA SPECIFIED
        JSR [SHIFT.LOOP]                ;GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
                                        ; PLACES RIGHT.  ON RETURN, WR 0 WILL
                                        ; HAVE BITS 0-7 IN BITS 24-31, OTHER
                                        ; BITS ARE CLEAR
        BIS WR[0] TO LS[T6.SUB]         ;NOW LS 6 CONTAINS DATA TO WRITE IN
                                        ; TB IN CORRECT FORMAT  I.E. PA BITS
                                        ; IN BITS 00-14 AND BYTE OFFSET, MODIFY
                                        ; PROT A THROUGH PROT D, AND VALID IN
                                        ; BITS 25-31.

        MEM.REQ[WRITE.TB] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO WRITE
                                        ; THE TB
        WRITE.MEM LS[T6.SUB]            ;WRITE DATA FROM LS 6 IN TB
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT

;
;       SUBROUTINE TO SHIFT WR 0 TO THE RIGHT 8 PLACES
;
SHIFT.LOOP:
        MOV LS[#8] TO WR[1]             ;COUNTER IN WR 1 TO SHIFT 8 PLACES
SHIFT.LOOP.1:
        ROR WR[0]                       ;SHIFT WR 0 ONE PLACE RIGHT
        DEC WR[1],                      ;DECREMENT COUNTER
         DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
        JMP.IF[NEQ] TO [SHIFT.LOOP.1]   ;REPEAT UNTIL 8 SHIFTS HAVE OCCURRED
        RETURN                          ;THEN RETURN
;
;       SUBROUTINE TO POSITION TB ADDRESS IN BITS 9-14 AND BIT 31 (TB ADDRESS
;       BIT 0 MUST BE IN BIT 31).
;
SHIFT.TB.ADR:
        MOV LS[T5.SUB] TO WR[0]         ;GET TB ADDRESS FROM LS 5
        BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE (8 BIT ADDRESS)
        SHL2 WR[0]                      ;SHIFT 2 PLACES LEFT TO POSITION ADDRESS
        SHL2 WR[0]                      ;SHIFT 2 PLACES LEFT TO POSITION ADDRESS
        SHL2 WR[0]                      ;SHIFT 2 PLACES LEFT TO POSITION ADDRESS
        SHL2 WR[0]                      ;SHIFT 2 PLACES LEFT TO POSITION ADDRESS
        ASHL WR[0]                      ;SHIFT ONE MORE PLACE
                                        ; NOW BITS 0-6 IN BITS 9-15
        BIT LS[BIT15] WITH WR[0],       ;SEE IF BIT 15 (MSB) IS SET OR CLEAR.
         DT(LONG)&SET.ALU.CC            ; MSB MUST GO IN BIT 31 IN ORDER TO AD-
                                        ; DRESS TB CORRECTLY
        SKIP.IF[EQL]                    ;SKIP NEXT INSTRUCTION IF MSB IS 0
        BIS LS[BIT31] TO WR[0]          ;ELSE SET BIT 31 FOR MSB
```

```
	MOV WR[0] TO LS[T5.SUB],	;NOW TB ADDRESS IS IN CORRECT FORM IN
```

```
                                         ; LS LOCATION 5
         RETURN                          ; THEN RETURN TO MAIN CODE
.PAGE   "  EXAMINE TRANSLATION BUFFER ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       This routine will read the translation buffer of the memory controller.
;       The TB will be returned with the byte offset, modify, prot A through
;       prot D, and valid in bits 01-07 respectively.  PA bits 09-23 are re-
;       turned in bits 08-22 respectively.  Bits 23-31 and bit 0 are not used
;       and so will be cleared before printing the result.  The result will be
;       put in LS 6 for the console to print.
;         The routine will issue a memory request with memory function=READ.TB
;       and data type=LONGWORD.  The console will load LS 5 with the address
;       specified before executing this routine.  The routine will shift the
;       address around as required to address the TB location specified.  The
;       address specified must be in HEX format.
;         The TB consists of 128 decimal locations (addresses from 0-7F).  The
;       remaining locations in the TB RAM are either unused or used by the
;       UNIBUS map.  UNIBUS map addresses are read via the EXAMINE UB routine.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 53) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS 5 - TB address (0-7F) in hex format
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       LS 6 - TB data from address specified
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       LS 5 is modified
;       WR 0 is modified
;
;---

EXAMINE.TB:
        JSR [SHIFT.TB.ADR]               ;SHIFT ADDRESS SPECIFIED FROM LS 5 INTO
                                         ; CORRECT POSITION AND REPLACE IN LS 5
        MEM.REQ[READ.TB] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO READ
                                         ; THE TB AT ADDRESS CONTAINED IN LS 5
        MOV MEM.DATA TO WR[0]            ;PLACE RESULT IN WR 0
```

```
	BIC LS[#FF000000] TO WR[0]	;CLEAR UPPER BYTE OF RESULT (UPPER BYTE
                                        ; IS NOT PART OF TB AND IS UNPREDICT-
                                        ; ABLE)
        BIC LS[BIT23] TO WR[0]          ;DITTO FOR BIT 23
        BIC LS[BIT0] TO WR[0]           ;DITTO FOR BIT 0.  WR 0 NOW CONTAINS
                                        ; RESULT TO PRINT
        MOV WR[0] TO LS[T6.SUB]         ;RESULT NOW IN LS 6
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
.PAGE   "  DEPOSIT UNIBUS MAP ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       This routine allows a write to the UNIBUS map portion of the TB on the
;       memory controller.  The map area contains 512 decimal locations
;       addressed from 200-3FF.  The remainder of the TB RAM is either unused
;       or contains the Translation Buffer information.  The DEPOSIT TB command
;       is used to write the TB portion of the TB RAMs.  The address specified
;       with the DEPOSIT command will be the actual RAM address accessed.  This
;       routine will do any shifting necessary on the address specified to pos-
;       ition it correctly.  The address specified has been loaded by the con-
;       sole in LS location 5 prior to executing this routine.  The address
;       must be in HEX format.
;         The data specified has been placed in LS 6 by the console prior to
;       executing this routine.  Bits 07-01 will be the Valid, Prot, Modify,
;       and Byte Offset bits (there are four Prot bits).  Bits 22-08 will
;       contain the PA bits from PA 23 through PA 09 respectively.  Bits 31
;       through 23 and bit 0 are unused. This routine will do any shifting nec-
;       essary to write the bits in TB as described.  The data specified must
;       be in HEX format.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 58) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS 5 - UNIBUS Map address in TB (must be in range 200-3FF HEX).
;       LS 6 - Contains the data to write into the UNIBUS Map in bits 1-22.
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       The UNIBUS map portion of the TB RAMs gets written with the data from
;       LS 6 at the address specified in LS 5.
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       WR 0 will be modified
;       WR 1 will be modified
```

```
;
```

```
;---

DEPOSIT.UBS:
        JSR [SHIFT.UB.ADR]              ;GO TO SUBROUTINE TO ARRANGE UNIBUS
                                        ; MAP ADDRESS CORRECTLY
        MOV LS[T6.SUB] TO WR[0]         ;GET DATA FROM LS 6
        BIC LS[#FF] TO WR[0]            ;CLEAR LOW BYTE OF DATA SPECIFIED.
                                        ; LOW BYTE WILL BE PUT IN BITS 24-31
                                        ; LATER
        JSR [SHIFT.LOOP]                ;GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
                                        ; PLACES RIGHT
        SWAP LS[T6.SUB] WITH WR[0]      ;SWAP ORIGINAL DATA SPECIFIED WITH MOD-
                                        ; IFIED DATA.  NOW LS 6 CONTAINS BITS
                                        ; 8-22 IN BITS 0-14 AS REQUIRED.  WR 0
                                        ; CONTAINS ORIGINAL DATA SPECIFIED
        BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE OF ORIGINAL
                                        ; DATA SPECIFIED
        JSR [SHIFT.LOOP]                ;GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
                                        ; PLACES RIGHT.  ON RETURN, WR 0 WILL
                                        ; HAVE BITS 0-7 IN BITS 24-31, OTHER
                                        ; BITS ARE CLEAR
        BIS WR[0] TO LS[T6.SUB]         ;NOW LS 6 CONTAINS DATA TO WRITE IN
                                        ; TB IN CORRECT FORMAT  I.E. PA BITS
                                        ; IN BITS 00-14 AND BYTE OFFSET, MODIFY
                                        ; PROT A THROUGH PROT D, AND VALID IN
                                        ; BITS 25-31.

        MEM.REQ[WRITE.UBS.MAP] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                                        ; WRITE THE UNIBUS MAP
        WRITE.MEM LS[T6.SUB]            ;WRITE DATA FROM LS 6 IN UNIBUS MAP
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
;
;       SUBROUTINE TO POSITION ADDRESS SPECIFIED CORRECTLY TO ADDRESS THE
;       UNIBUS MAP.
;
SHIFT.UB.ADR:
        MOV LS[T5.SUB] TO WR[0]         ;GET ADDRESS SPECIFIED IN WR 0
        SHL2 WR[0]                      ;SHIFT ADDRESS 2 PLACES LEFT
        SHL2 WR[0]                      ;SHIFT ADDRESS 2 PLACES LEFT
        SHL2 WR[0]                      ;SHIFT ADDRESS 2 PLACES LEFT
        SHL2 WR[0]                      ;SHIFT ADDRESS 2 PLACES LEFT
        ASHL WR[0]                      ;SHIFT LEFT ONE MORE PLACE
                                        ; NOW ADDRESS IS IN BITS 9-17 OF WR 0
        MOV WR[0] TO LS[T5.SUB],        ;NOW LS 5 CONTAINS CORRECT ADDRESS.
         RETURN                         ; RETURN TO MAIN

.PAGE   "  EXAMINE UNIBUS MAP ROUTINE"
;+++
;       This routine will read the UNIBUS map portion of the memory controller.
;       The UNIBUS MAP contents will be returned with the byte offset, modify,
;       prot A through prot D, and valid in bits 01-07 respectively.  PA bits
;       09-23 are returned in bits 08-22 respectively.  Bits 23-31 and bit 0
;       are not used and so will be cleared before printing the result.  The
;       result will be put in LS 6 for the console to print.
;         The routine will issue a memory request with memory function=READ.UBS
;       .MAP and data type=LONGWORD.  The console will load LS 5 with the ad-
;       dress specified before executing this routine.  The routine will shift
;       the address around as required to address the UNIBUS map location spec-
;       ified.  The address specified must be in HEX format.
;         The UNIBUS map consists of 512 decimal locations (addresses from 200-
;       3FF)  The remaining locations in the TB RAM are either unused or used
;       by the translation buffer.  TB contents are read via the EXAMINE TB
;       routine.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 59) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS 5 - TB address in hex format
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       LS 6 - UNIBUS map data from address specified
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       LS 5 is modified
;       WR 0 will be modified
;
;---

EXAMINE.UBS:
        JSR [SHIFT.UB.ADR]              ;SHIFT ADDRESS SPECIFIED FROM LS 5 INTO
                                        ; CORRECT POSITION AND REPLACE IN LS 5
        MEM.REQ[READ.UBS.MAP] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                                        ; READ THE UBS MAP AT ADDRESS CONTAINED
                                        ; IN LS 5
        MOV MEM.DATA TO WR[0]           ;PLACE RESULT IN WR 0

        BIC LS[#FF000000] TO WR[0]      ;CLEAR UPPER BYTE OF RESULT (UPPER BYTE
                                        ; IS NOT PART OF TB AND IS UNPREDICT-
                                        ; ABLE)
        BIC LS[BIT23] TO WR[0]          ;DITTO FOR BIT 23
        BIC LS[BIT0] TO WR[0]           ;DITTO FOR BIT 0.  WR 0 NOW CONTAINS
                                        ; RESULT TO PRINT
        MOV WR[0] TO LS[T6.SUB]         ;RESULT NOW IN LS 6
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
                                        ; CONTROL
.PAGE   "  DEPOSIT MAIN MEMORY ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
```

```
;       This routine write a longword of data into main memory at the Physical
;       address specified.  This address need not be on a longword boundary.
;       The console will use this routine for data transfers to main memory as
;       well as for the DEPOSIT MM command.  The console will have loaded the
;       address into LS 5 and the data into LS 6 before executing this routine.
;         The routine issues a memory request with the memory function=WRITE.P
;       and the data type=longword.  It then issues a WRITE.MEM LS[6] instruc-
;       tion to write the data into memory.  A check on ERROR SUM is made to
;       assure that the write occured without an error.  If ERROR SUM is as-
;       serted, the CPU will issue a CPU ACK before asserting CPU ATTN to in-
;       form the console that an error occurred.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 54) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS 5 - Main memory physical address
;       LS 6 - Main memory longword data
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       Main Memory address specified is written with data from LS 6.
;
; IMPLICIT OUTPUTS:
;
;       A memory error asserts CPU ACK to inform the console of a write error.
;
; SIDE EFFECTS:
;
;       None
;
;---

DEPOSIT.MM:
        MEM.REQ[WRITE.P] ADRS[T5.SUB] DT[LONG]  ;SET UP FOR WRITE TO MAIN
                                        ; MEMORY USING THE PHYSICAL ADDRESS
                                        ; STORED IN LS 5
        WRITE.MEM LS[T6.SUB],           ;WRITE THE DATA FROM LS 6 INTO MAIN
                                        ; MEMORY
         SKIP.IF[MEM.REF.OK]            ;SKIP NEXT INSTRUCTION IF NO MEMORY
                                        ; ERROR (NO ERR SUM)
        MISC [SET.CP.ACK]               ;SET CPU ACK IF ERROR SUM ASSERTED

        MOV LS[#4] TO WR[0]             ;PUT A 4 IN WR 0
        ADD WR[0] TO LS[T5.SUB]         ;INCREMENT LS 5 FOR NEXT LONGWORD DEP
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
.PAGE   "  EXAMINE MAIN MEMORY ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
```

```
;       This routine is used to read a longword form main memory at the Phy-
;       sical address specified.  The address need not be on a longword bound-
;       ary.  This routine is used by the console on an EXAMINE.MM command.
;       The console will have loaded LS 5 with the physical address specified
;       before executing this routine.
;         The routine will first write CSR 1 to set the INH REP CRD (inhibit
;       report correctable read data) bit to prevent an error sum from occuring
;       on a single bit error which has been corrected.  Then the routine is-
;       sues a memory request with memory function=READ.P and DT=longword.  It
;       then reads the memory location and puts the result in LS 6.  It also
;       checks for an ERROR SUM and issues a CPU ACK if an error occurred (sin-
;       gle bit errors that have been corrected will not cause an ERR SUM).  If
;       an error occurred, CPU ACK will be set before issueing CPU ATTN to in-
;       form the console that an error occurred.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 55) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS 5 - Physical address in main memory to read.
;
; IMPLICIT INPUTS:
;
;       LS[INH.CRD] - Data to write into CSR 1 to inhibit error sum on a CRD.
;       LS[CSR1]  - Address for writing CSR 1.
;
; OUTPUT PARAMATERS:
;
;       LS 6 - Longword data from physical memory address specified.
;
; IMPLICIT OUTPUTS:
;
;       CPU ACK if an error sum occurs.
;
; SIDE EFFECTS:
;
;       None
;
;---

EXAMINE.MM:
        MEM.REQ[WRITE.CSR] ADRS[CSR1] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                                        ; WRITE CSR 1
        WRITE.MEM LS[INH.CRD]           ;SET INH REP CRD IN CSR 1 AND CLEAR
                                        ; ECC DISABLE

        MEM.REQ[READ.P] ADRS[T5.SUB] DT[LONG]   ;ISSUE MEMORY REQUEST TO
                                        ; READ LONGWORD DATA FROM MAIN MEMORY
                                        ; PHYSICAL ADDRESS IN LS 5
        MOV MEM.DATA TO LS[T6.SUB],     ;READ DATA AND PUT IN LS 6
         SKIP.IF[MEM.REF.OK]            ; SKIP NEXT INSTRUCTION IF NO MEMORY
                                        ; ERROR (NO ERROR SUM)
        MISC [SET.CP.ACK]               ;SET CPU ACK IF AN ERROR OCCURRED
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
.page   " EXAMINE IDC ROUTINES"
```

```
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       The following routines are used to read data from the optional IDC
;       module's registers.  The result is put in LS 6 for the console to
;       print out.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 5A through 5E) into the UPC and starts the
;       CPU.
;
; INPUT PARAMETERS:
;
;       None
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       LS 6 - Data from IDC register specified.
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       None
;
;---

EXAMINE.ICSR:
        MISC [SET.PORT.I.O]                 ;SELECT PORT TO BUS
        PORT.READ PORT.ADRS[CSR]            ;SEND THE READ COMMAND
        JMP [READ.IDC]                      ;READ THE DATA

EXAMINE.IDAR:
        MISC [SET.PORT.I.O]                 ;SELECT PORT TO BUS
        PORT.READ PORT.ADRS[DAR]            ;SEND THE READ COMMAND
        JMP [READ.IDC]                      ;READ THE DATA

EXAMINE.DBUF:
        MISC [SET.PORT.I.O]                 ;SELECT PORT TO BUS
        PORT.READ PORT.ADRS[DATA.WORD]      ;SEND THE READ COMMAND
        JMP [READ.IDC]                      ;READ THE DATA

EXAMINE.PATT:
        MISC [SET.PORT.I.O]                 ;SELECT PORT TO BUS
        PORT.READ PORT.ADRS[PATTERN]        ;SEND THE READ COMMAND
        JMP [READ.IDC]                      ;READ THE DATA

EXAMINE.POSIT:
        MISC [SET.PORT.I.O]                 ;SELECT PORT TO BUS
        PORT.READ PORT.ADRS[POSITION]    ;SEND THE READ COMMAND
```

```
READ.IDC:
        MOV PORT TO LS[T6.SUB]            ;READ DATA AND PUT IN LS 6
        MISC [SET.ACC.I.O]                ;RETURN SELECT TO ACCELERATOR
        JMP [WAIT.SUB]                    ;JUMP TO SET ATTN AND WAIT
.page   "  DEPOSIT IDC ROUTINES"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       The following routines are used to write data to the optional IDC
;       module's registers.  The data is taken from LS 6 which is previously
;       loaded by the monitor when the DEPOSIT command is executed.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 5F through 61) into the UPC and starts the
;       CPU.
;
; INPUT PARAMETERS:
;
;       LS 6 - Data pattern to write.
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       None
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       None
;
;---

DEPOSIT.ICSR:
        MOV LS[T6.SUB] TO WR[0]           ;GET DATA PATTERN TO WRITE
        PORT.WRITE PORT.ADRS[CSR] WITH WR[0]    ;WRITE TO IDC
        JMP [WAIT.SUB]                    ;JUMP TO SET ATTN AND WAIT


DEPOSIT.IDAR:
        MOV LS[T6.SUB] TO WR[0]           ;GET DATA PATTERN TO WRITE
        PORT.WRITE PORT.ADRS[DAR] WITH WR[0]    ;WRITE TO IDC
        JMP [WAIT.SUB]                    ;JUMP TO SET ATTN AND WAIT


DEPOSIT.DBUF:
        MOV LS[T6.SUB] TO WR[0]           ;GET DATA PATTERN TO WRITE
```

```
	PORT.WRITE PORT.ADRS[DATA.WORD] WITH WR[0]	;WRITE TO IDC
```

```
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT


CLEAR.FIFO.ADDR:
        PORT.CONTROL [CLEAR.FIFO.CNTR]  ;CLEAR ADDRESS IN FIFO A OR FIFO B
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT


SELECT.FIFO.A:
        PORT.CONTROL [SEL.FIFO.A]       ;SELECT FIFO A
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT


SELECT.FIFO.B:
        PORT.CONTROL [SEL.FIFO.B]       ;SELECT FIFO B

WAIT.SUB:
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.DE.EX:
        JMP [WAIT.DE.EX]                ;LOOP SELF UNTIL CONSOLE TAKES CONTROL
.PAGE   "  SAVE WR ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       This routine saves WRs 0-3 in LS locations 1-4.  When the console takes
;       control of the CPU, it calls this routine so that the working registers
;       can be used by the console.  The WRs are restored by another routine
;       called by the console before the CPU resumes execution (on a COntinue
;       command, for example).
;         This routine simply moves data from WR 0-3 to LS locations 1-4 and
;       then issues a CPU ATTN to the console.  It then loops on a JMP self
;       until the console takes over.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 46) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       None
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       LS 1 - Contents of WR 0
;       LS 2 - Contents of WR 1
;       LS 3 - Contents of WR 2
;       LS 4 - Contents of WR 3
;
; IMPLICIT OUTPUTS:
;
;       None
```

```
;
```

```
; SIDE EFFECTS:
;
;       None
;
;---

SAVE.WR:

        MOV WR[0] TO LS[SAV.WR0]        ;SAVE WR 0 IN LS LOCATION 1
        MOV WR[1] TO LS[SAV.WR1]        ;SAVE WR 1 IN LS LOCATION 2
        MOV WR[2] TO LS[SAV.WR2]        ;SAVE WR 2 IN LS LOCATION 3
        MOV WR[3] TO LS[SAV.WR3]        ;SAVE WR 3 IN LS LOCATION 4
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
.PAGE   "  RESTORE WR ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       This routine restores the 2901 working registers save in LS 1-4.  The
;       console calls this routine before the CPU is restarted after it has
;       been halted by the console.
;         The routine simply moves the data from LS locations 1-4 to WRs 0-3
;       and then issues a CPU ATTN to the console.  A JMP self is executed next
;       waiting for the console to take control.
;
; CALLING SEQUENCE:
;
;       The console loads the address of a pointer to this routine (a JMP in-
;       struction at WCS location 47) into the UPC and starts the CPU.
;
; INPUT PARAMETERS:
;
;       LS 1 contains the data to be restored in WR 0.
;       LS 2 contains the data to be restored in WR 1.
;       LS 3 contains the data to be restored in WR 2.
;       LS 4 contains the data to be restored in WR 3.
;
; IMPLICIT INPUTS:
;
;       None
;
; OUTPUT PARAMATERS:
;
;       WR 0 gets data from LS 1.
;       WR 1 gets data from LS 2.
;       WR 2 gets data from LS 3.
;       WR 3 gets data from LS 4.
;
; IMPLICIT OUTPUTS:
;
;       None
;
; SIDE EFFECTS:
;
;       None
;
;---
```

```
RESTORE.WR:
        MOV LS[SAV.WR0] TO WR[0]        ;RESTORE WR 0
        MOV LS[SAV.WR1] TO WR[1]        ;RESTORE WR 1
        MOV LS[SAV.WR2] TO WR[2]        ;RESTORE WR 2
        MOV LS[SAV.WR3] TO WR[3]        ;RESTORE WR 3
        JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
.PAGE   "WCS SUBROUTINES FOR CPU USE"
.TOC    "  ERROR ROUTINE"
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       This routine is used to detect and report diagnostic errors occurring
;       during WCS based micro-diagnostics.  The WCS micro-code sets up the
;       parameters listed below and calls this routine.  The routine compares
;       WR 1 (expected data) and WR 0 (received data) according to the error
;       mask.  Bits set in the mask indicate bits that are not checked for
;       errors.
;        If WR 0 and WR 1 are equal (no error) the routine executes a return+1
;       to the calling point.  The only exception to this is if loop on error
;       is set.  In that case, the error number is checked to see if that error
;       occurred previously.  If so, then the error loop is forced by doing a
;       return.  This will lock an error loop in on intermittent errors.
;        If WR 0 and WR 1 are not equal (an error) then bit 8 is set in the
;       CONTROL AND STATUS word, expected and received data is set up in LS for
;       printout, and flags are checked in the ERROR CONTROL word.  This con-
;       trols whether printouts are disabled, loop on error is enabled, etc.
;       After printing the error, a return+1 is made to the calling point.
;        Loop on error causes a return rather than return+1 to cause an error
;       loop.  Also CPU ACKNOWLEDGE is toggled at the end of the routine for
;       a scope sync point.
;
; CALLING SEQUENCE:
;
;       JSR [CHECK.RESULT]
;
; INPUT PARAMETERS:
;
;       WR[0] = Received data i.e. the actual result of the test
;       WR[1] = Expected data i.e. what the result should have been
;
; IMPLICIT INPUTS:
;
;       LS[ERROR.MASK] = Used to mask out bits that are not to be tested
;       LS[ERROR.NUMBER] = Current error number
;       LS[PREVIOUS.ERROR] = Error number of previous data
;       LS[CONTROL.STATUS] = Control and Status word.  Contains information
;           written by the CPU to inform the monitor what to do.
;       LS[ERROR.CONTROL] = Information such as loop-on-error, bell-on-error,
;           no error report, and halt on error.
;
; OUTPUT PARAMATERS:
;
;       NONE
;
; IMPLICIT OUTPUTS:
;
```

```
;       LS[CONTROL.STATUS] = Control and Status with new status information
;       LS[PREVIOUS.ERROR] = Last error number (modified only if error and
;           loop on error is enabled)
;       LS[EXPECTED.DATA] = Expected result if an error was detected
;       LS[RECEIVED.DATA] = Actual result if an error was detected
;
; SIDE EFFECTS:
;
;       WR[0], WR[1], and the Q reg are modified and are not restored before
;           returning.
;       If loop-on-error is enabled and an error loop is to be executed, CPU
;           ACKNOWLEDGE will be toggled for a scope sync.
;
;---

CHECK.RESULT:
        BIC LS[ERROR.MASK] TO WR[0]     ;MASK OFF NOT TESTED BITS (CLEAR THEM)
                                        ; FOR RECEIVED DATA
        BIC LS[ERROR.MASK] TO WR[1]     ;DITTO FOR EXPECTED DATA

        XOR WR[0] WITH WR[1] TO Q,      ;CMP RECEIVED DATA IN WR[0] WITH
         DT(LONG)&SET.ALU.CC            ; EXPECTED DATA IN WR[1] FOR ERROR
        JMP.IF[NEQ] TO [ERROR]          ;JUMP IF ERROR

        MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONTROL WORD FROM LS
        BIT WR[0] WITH LS[LOE],         ;SEE IF LOOP ON ERROR IS ENABLED
         DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
        SKIP.IF[BIT.SET]                ;SKIP IF IT IS
        RETURN+1                        ;ELSE RETURN+1 (NO ERROR AND NO LOOP)

        MOV LS[PREVIOUS.ERROR] TO WR[0] ;IF NO ERROR BUT LOOP ON ERROR IS
                                        ; ENABLED, SEE IF THERE WAS A
                                        ; PREVIOUS ERROR
        XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,
         DT(LONG)&SET.ALU.CC            ;IS THIS THE SAME SPOT THAT HAD AN
                                        ; ERROR BEFORE? (INTERMITTANT ERROR)
        JMP.IF[NEQ] TO [RET+1]          ;JUMP IF NOT TO RETURN WITHOUT ERROR
                                        ; LOOPING (RETURN+1)
        JMP [RET]                       ;ELSE CONTINUE TO LOOP (WE WILL LOOP
                                        ; ON ERROR EVEN IF IT GOES AWAY)

ERROR:
        MOV WR[0] TO LS[RECEIVED.DATA]  ;PUT RESULT OF TEST IN LS FOR PRINTOUT
        MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONROL BITS TO CHECK FOR
                                        ; LOOP COMMAND
        BIT WR[0] WITH LS[LOOP.COMMAND],;SEE IF BIT 6 SET
         DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
        JMP.IF[BIT.SET] TO [RET]        ;GO LOOP IF SET (FAST LOOP)

        MOV WR[1] TO LS[EXPECTED.DATA]  ;PUT EXPECTED DATA IN LS FOR PRINTOUT

        MOV LS[CONTROL.STATUS] TO WR[1] ;GET CONTROL AND STATUS WORD FROM LS
        BIC LS[#FE7FFFFF] TO WR[1]      ;CLEAR ALL BUT BITS 23&24 IN CONTROL
                                        ; AND STATUS WORD
        BIS LS[ERROR] TO WR[1]          ;SET BIT 8 IN CONTROL AND STATUS WORD
                                        ; (INDICATES AN ERROR WAS DETECTED)
        MOV WR[1] TO LS[CONTROL.STATUS] ;WRITE UPDATED CONTROL AND STATUS WORD
                                        ; BACK TO LS[40]
```

```
        BIT WR[0] WITH LS[BELL],        ;SEE IF BELL ON ERROR IS SET (WR 0 STILL
                                        ; CONTAINS ERROR CONTROL WORD)
         DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
        JMP.IF[BITS.CLR] TO [CHECK.NER] ;IF BELL ON ERROR IS NOT SET, JUMP TO
                                        ; SEE IF ERROR REPORTING IS INHIBITED
        MISC [SET.CP.ATTN]              ;ELSE SET CPU ATTN (RING MY BELL)

WAIT.1: JMP [WAIT.1]                    ;WAIT FOR CONSOLE TO STOP ME
        JMP [LOOP]                      ;GO TO CHECK LOOP-ON-ERROR AFTER
                                        ; CONSOLE HAS FINISHED

CHECK.NER:
        BIT WR[0] WITH LS[NER],         ;IF NO BELL SEE IF ERROR REPORT IS
                                        ; INHIBITED
         DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
        JMP.IF[BIT.SET] TO [CHECK.HALT] ;JUMP IF ERROR REPORT IS INHIBITED TO
                                        ; CHECK FOR HALT ON ERROR

        MISC [SET.CP.ATTN]              ;SET CPU ATTN (REPORT ERROR)
WAIT.2: JMP [WAIT.2]                    ;WAIT FOR CONSOLE TO STOP ME
        JMP [LOOP]                      ;GO TO CHECK LOOP-ON-ERROR AFTER
                                        ; CONSOLE HAS FINISHED

CHECK.HALT:
        BIT WR[0] WITH LS[HOE],         ;SEE IF HALT ON ERROR IS ENABLED
         DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
        JMP.IF[BITS.CLR] TO [LOOP]      ;JUMP IF NOT TO CHECK LOOP-ON-ERROR

        MISC [SET.CP.ATTN]              ;ELSE SET CPU ATTN (HALT ON ERROR)
WAIT.3: JMP [WAIT.3]                    ;WAIT FOR CONSOLE TO STOP ME

LOOP:   MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONTROL BITS (NER, HOE ETC.)
        BIT WR[0] WITH LS[LOE],         ;SEE IF LOOP-ON-ERROR
         DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
        SKIP.IF[BIT.SET]                ; SKIP IF LOOP ON ERROR IS ENABLED

RET+1:  RETURN+1                        ;ELSE RETURN+1 FOR NO ERROR LOOP

        MOV LS[ERROR.NUMBER] TO WR[0]   ;GET ERROR NUMBER IF LOOPING

        MOV WR[0] TO LS[PREVIOUS.ERROR] ;AND SAVE IT IN PREVIOUS ERROR

RET:    MISC [SET.CP.ACK]               ;SET CPU ACKNOWLEDGE FOR SCOPE SYNC
        MISC [CLR.CP.ATTN.AND.ACK],     ;AND THEN CLEAR IT
         RETURN                         ; RETURN TO TEST AND LOOP ON ERROR

.PAGE
;
;
;
ERR.NO.TEST:
        NOP                             ;NOP WILL CAUSE A SEQUENCE ERROR
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                                        ; STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                                        ; BIT 15 INDICATES START OF TEST TO
                                        ; CONSOLE
```

```
	JMP [WAIT.SUB]				;JUMP TO SET ATTN AND WAIT
```

```
.TOC    "START TRANSFER ROUTINE"
;+++
;
;       This routine loads the LS and main memory with information neccesary
;       for these tests to run.  It will then issue a CPU ATTN with the control
;       and status word clear to indicate that all transfers are complete.
;
;---

TRANSFER.POINT:

        CLR WR[0]                        ;START WITH 0 IN WR[0]
        MOV WR[0] TO LS[PSL.HW]          ;MAKE SURE COMPAT MODE IS CLEAR

        CLR WR[0]                        ;START WITH 0 IN WR[0]
        INC WR[0]                        ;NOW HAVE 1 IN WR[0]
        ASHL WR[0]                       ;0 IN BIT 0, 1 IN BIT 1
        INC WR[0]                        ;11(B)
        ASHL WR[0]                       ;110(B)
        INC WR[0]                        ;111(B)
        ASHL WR[0]                       ;1110(B)
        ASHL WR[0]                       ;1 1100(B)
        ASHL WR[0]                       ;11 1000(B)
        ASHL WR[0]                       ;111 0000(B)  NOW HAVE 70(H) IN WR 0.
                                         ; THIS IS CORRECT START ADDRESS OF DATA
                                         ; SECTION (INCLUDING LONGWORD COUNT,
                                         ; DESTINATION, AND DESTINATION ADDRESS)
        MOV WR[0] TO LS[TRANSFER.POINTER]  ;LOAD LS 7 WITH POINTER TO TRANSFER
                                         ; DATA SECTION.

        CLR WR[0]                        ;0 IN WR 0
        INC WR[0]                        ;1 IN WR 0
        ASHL WR[0]                       ;10(B)
        ASHL WR[0]                       ;100(B)
        ASHL WR[0]                       ;1000(B)
        ASHL WR[0]                       ;1 0000(B)
        ASHL WR[0]                       ;10 0000(B)
        ASHL WR[0]                       ;100 0000(B)
        ASHL WR[0]                       ;1000 0000(B)
        ASHL WR[0]                       ;1 0000 0000(B)
        ASHL WR[0]                       ;10 0000 0000(B)
        ASHL WR[0]                       ;100 0000 0000(B)
        ASHL WR[0]                       ;1000 0000 0000(B)
        ASHL WR[0]                       ;1 0000 0000 0000(B)
        ASHL WR[0]                       ;10 0000 0000 0000(B)
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 13 IN LS CONTROL AND STATUS
                                         ; WORD (BIT 13 INDICATES CONSOLE MUST
                                         ; PERFORM A DATA TRANSFER)
        JSR [SET.ATTN]                   ;DO FIRST LS TRANSFER TO LOCATIONS 0
                                         ; THROUGH 77(H)


        MOV LS[#162(H)] TO WR[0]         ;GET START ADDRESS OF SECOND HALF OF LS
                                         ; TRANSFER
        MOV WR[0] TO LS[TRANSFER.POINTER] ;LOAD START ADDRESS IN LS FOR CONSOLE
        MOV LS[XFER.DATA] TO WR[0]       ;SET BIT 13 IN WR
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT TO DO DATA TRANSFER
```

```
	JSR [SET.ATTN]			;DO SECOND LS TRANSFER TO LOCATIONS 80
```

```
                                        ; THROUGH F7(H)


        MOV MEM.DATA TO WR[0]           ;THIS INSTRUCTION USED TO FREE MEMORY
                                        ; IF HUNG IN A OCTA FLOW
        MOV MEM.DATA TO WR[0]           ;NEEDS 4 OF THESE
        MOV MEM.DATA TO WR[0]
        MOV MEM.DATA TO WR[0]           ;DONE WITH MEMORY INITIALIZATION


        MOV LS[#254] TO WR[0]           ;GET START ADDRESS OF MAIN MEMORY
                                        ; TRANSFER DATA
        MOV WR[0] TO LS[TRANSFER.POINTER] ;LOAD START ADDRESS IN LS FOR CONSOLE
        JSR [SET.ATTN]                  ;DO MAIN MEMORY TRANSFER TO LOCATIONS
                                        ; SPECIFIED IN DATA FILE


        MEM.REQ[WRITE.CSR] ADRS[CSR1] DT[LONG]
        WRITE.MEM LS[ZERO]              ;DISABLE MEMORY MANAGEMENT

        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET IPL TO 1F

        PORT.CONTROL [CLEAR.IDC]        ;JAM IDC TO IDLE
        PORT.CONTROL [CLEAR.IDC]
        PORT.CONTROL [CLEAR.IDC]
        PORT.CONTROL [CLEAR.IDC]        ;DONE JAMMING IDC

        MOV LS[#80] TO WR[1]            ;SET BIT 7 OF WR 1
        PORT.WRITE PORT.ADRS[CSR] WITH WR[1] ; CLEAR CRDY IN IDC
        PORT.CONTROL [CLEAR.AUTO]       ;CLEAR AUTO MODE IN IDC
        PORT.CONTROL [RESET.BR]         ;CLEAR BR 7 IN IDC

        MOV LS[BIT16] TO WR[0]
        BIS LS[UBS.DCLO] TO WR[0]
        MOV WR[0] TO LS[CONTROL.STATUS] ;CLEAR ALL INTERRUPTS BY ISSUEING DC LO
        JSR [SET.ATTN]                  ;ISSUE CPU ATTN AND WAIT FOR RESTART

        CLR LS[CONTROL.STATUS]
        JMP [SET.ATTN]                  ;ISSUE CPU ATTN WITH CONTROL AND STATUS
                                        ; WORD CLEAR
.LIST
.PAGE
.REGION/700,3FFF
700:
        INC WR[3]                       ;BLOCK OF 256 INCREMENTS
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                       ;
        INC WR[3]                      ;       |
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
INC WR[3]                        ;
```

```
	INC WR[3]			;		|
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
```

```
        INC WR[3]                              ;          |
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
INC WR[3]                       ;
```

```
     INC WR[3]                         ;         |
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
	INC WR[3]				;
```

```
        INC WR[3]                         ;        |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ;         |
        INC WR[3]                       ; --------V--------
        RETURN
    .Page "General Use Subroutines"
;This subroutine asserts the N/A flag for expected and received data fields.
810:
ASSERT.NA:
        MOV LS[BIT23] TO WR[0]
        BIS WR[0] TO LS[CONTROL.STATUS] ;CSR = BIT23
        RETURN
;
;This subroutine deasserts the N/A flag for expected and received data fields.
DEASSERT.NA:
        MOV LS[CONTROL.STATUS] TO WR[0]
        BIC LS[BIT23] TO WR[0]
        MOV WR[0] TO LS[CONTROL.STATUS]
        RETURN
;
;This subroutine asserts the flag for "other" data field.
ASSERT.OTHER:
        MOV LS[BIT24] TO WR[0]
        BIS WR[0] TO LS[CONTROL.STATUS]
        RETURN
;
;This subroutine deasserts the flag for "other" data field.
DEASSERT.OTHER:
        MOV LS[CONTROL.STATUS] TO WR[0]
        BIC LS[BIT24] TO WR[0]
        MOV WR[0] TO LS[CONTROL.STATUS]
        RETURN

;
;This subroutine issues CPU ATTENTION and waits for the console to restart
;the CPU.

SET.ATTN:
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.XFER:
        JMP [WAIT.XFER]                 ;LOOP HERE WAITING FOR CONSOLE TO RE-
                                        ; SPOND
        RETURN                          ;BACK TO MAIN

    .Page "TEST 1 - IB BUS and Latches Test(M8390 CPU MODULE)"
    ;++
    ;
    ;
    ; Test Description:
    ;
```

```
;	This test checks the IB BUS and registers of the CPU module.  A
```

```
;       memory request with IB.FILL is issued at the beginning and after every
;       four byte checks to insure that a new 32 bit pattern is written
;       into the 4 IB latches in case the decode fails to issue an IB FILL
;       memory request.  Each byte will be individually enabled for data
;       checking.  The data path is as follows: bus MC D00 to D31 to four
;       latches forming BUS IB D00 to D07. Only one of these latches will
;       output on BUS IB D00 - D07 at any one time. The latch which outputs
;       is determined by the low 2 bits of the Y BUS at the beginning of the
;       decode instruction. The decode causes the OS bits OS 0 H through
;       OS 7 H to be loaded with BUS IB 0 H through BUS IB 7 H. These OS bits
;       are fed through 3 muxes and output on the D BUS, where they are
;       checked in the 2901. Only 8 bits are checked at a time. The other
;       bits are masked out. The expected data will be created by shifting
;       and complementing a 32 bit pattern. To check each byte the
;       lower 8 bits are used and shifted 8 times to get the next byte of
;       expected data. This test also checks that, MOV IB.DATA TO OS skips
;       on IB.VALID and reports an error if there's no skip. LS[10] contains
;       the address of the longword of memory to be moved into IB Prefetch.
;       The test will also check that LS[10] (macro PC) is incremented by
;       the decode instruction.  The data patterns used are ones,
;       zeros, shifting ones and shifting zeros.
;         The second part of this test checks the decode instruction which uses
;       the same data path as the first part of this test. Since the data path
;       and IB Prefetch register have been found to be working it is not
;       necessary to test with all data patterns. All ones and all zeros
;       will be used as patterns. With a MEM.REQ [IB.FILL] load the IB
;       Prefetch register with zeros, then using the decode instruction
;       keep loading the OS with bytes from the IB Prefetch register until
;       PC=3 and a memory request is initiated.
;
; Assumptions:
;
;       It is assumed that the memory controller and array micro-test has run
;       successfully and that memory data is valid.  However the LOAD IB
;       signal generated by the memory controller via a memory IB FILL request
;       has not been previously tested and is subject to causing an error.  It
;       is also assumed that the OS input to the MUXes which output on the D
;       BUS and the MUXes themselves are good.
;
; Test Steps:
;
;       1. Initialize registers and local store locations.
;       2. Execute a MEM.REQ [IB.FILL] to load the IB prefetch register.
;       3. Load a byte of IB Prefetch into OS and skip if IB Valid. If IB
;          Valid is not asserted then report error 1.
;       4. Load expected data from memory into WR[1].
;       5. Check data in OS.  If incorrect, report error 2.
;       6. Increment the total counter and check to see if LS[10] (macro
;          PC) has incremented.  Report error 3 if not.
;       7. Repeat steps 2 through 6 until all data patterns have been tested.
;       8. Fill IB with zeros via a MEM.REQ [IB.FILL].
;       9. Check four bytes of IB using the decode instruction MOV IB.DATA TO
;          OS.
;       10.Read a byte of new data from IB to check that decode initiated a
;          memory request when PC=3 in step 9.
;       11.Clear IB Valid, and check that decode does not cause skip anyway.
;       12.With IB Valid clear and no skip on IB Valid, check that decode does
```

```
  ;       not cause skip anyway.
```

```
;
; Error:
;
;     Error 1 - Test failed to skip on IB Valid.
;     Error 2 - Test data failed to go from memory to OS corectly using the
;               MEM.REQ [IB.FILL] instruction.
;     Error 3 - LS[10] (macro PC) failed to increment correctly on a
;               DECODE instruction.
;     Error 4 - Test data failed to go from memory to IB correctly using the
;               MEM.REQ [IB.FILL] instruction.
;     Error 5 - Decode instruction failed to correctly initiate a memory
;               request when PC=3.
;     Error 6 - Test skipped when IB Valid was not set.
;     Error 7 - Test skipped when State Flag 0 was clear.
;
; Debug:
;
;     Error 1: There are 3 possible causes for not skipping on IB Valid.
;             One is that the skip logic for the IB Valid is not working
;             correctly. This could occur in the SEQ.CTL PAL or the IB
;             VALID L input to this PAL. The second cause is that IB
;             VALID is not being asserted.  The IB.IR Control PAL or it's
;             inputs is suspect in this case. In these two cases the IB
;             Prefetch Data may be OK. The third possible cause is that the
;             memory is not returning LOAD IB H after a MEM.REQ with IB.FILL
;             Either the memory is not executing the IB.FILL function
;             correctly, or LOAD IB H is not making it back to the CPU
;             module. It is also possible that if the jump control field
;             of the decode instruction is incorrectly interpreted in the
;             SEQ.CTL PAL. It's possible that if a jump is enabled the jump
;             could take the program anywhere.
;     Error 2: This error is most likely caused by problems in the Data Path
;             After the machine has halted on error, the prefetch register
;             can be scoped to see if it is correct.  An error here indi-
;             cates that either the LOAD PREFETCH H signal is not clocking
;             the latch or, the data is not getting to the latch, or the
;             latch is inoperative.  LOAD PREFETCH H is produced by the
;             LOAD IB H signal from the memory controller.  The data lines
;             to the latch could be open or the memory may not be executing
;             an IB.FILL. The latch itself could be faulty, in which case
;             probably one byte of four is failing. Examine LS[10] with a
;             scope. The low two bits determine the byte that failed.
;             IF the prefetch register is OK, the problem may be in the
;             OS register or it's control. The two OS PAL's are OS<7:1><odd>
;             and OS<6:0><even>. Either one may be defective. It's control
;             comes from the CM.RDEST PAL. Both OS CTL 0 and OS CTL 1 should
;             be low when the clock registers signal is asserted. Another
;             problem that could occur is that the ENABLE IB signals 0
;             through 3 are not working correctly. This is controled by the
;             IB.IR CONTROL PAL and the decoder IC. The enable on the
;             decoder is asserted on the DECODE instruction when IB.REQ is
;             set. The data path after the OS PALS has been tested pre-
;             viously.
;     Error 3: There are two sources of possible error if LS[10] isn't
;             incremented. One is that the locations selected either in
;             the DEST.CTL ROM or the SRC.ALU.CTL PAL are bad causing the
;             source, destination, or function being sent to the 2901 to be
;               wrong. By single stepping it's possible to observe whether
```

```
    ;           the control signals are in error or not. The source should
    ;           be 7, the destination should be 1, the function should be
    ;           0, and the carry bit should be set. If these aren't correct
    ;           then the error is probably a bad location in ROM or a bad
    ;           PAL. If the control signals are correct then it's likely that
    ;           DAPA ENABLE BYTE isn't right. DAPA ENABLE BYTE comes from
    ;           PAL 200.
    ;    Error 4: This error most likely indicates that a memory referance is
    ;           being initiated before PC=3 and is changing the data in the
    ;           IB latches.  Check the signal PC=3 at the IB.IR CONTROL PAL
    ;           to make sure that it is not going high before the last DECODE
    ;           instruction is executed.
    ;    Error 5:  If a decode instruction with IB.Request set and PC=3 is
    ;            not initiating a memory request, then the probable source
    ;            of error is the control logic which is primarily located
    ;            in the IB.IR CONTROL PAL. The inputs to check are BUS Y
    ;            D00 H, BUS Y D01 H, DAPA DECODE INSTR H, and WCSH CSR 08
    ;            H. These inputs should all be high. If the inputs are
    ;            correct then the PAL could be in error. In this case the
    ;            output DAPF PC EQUALS 3 H would be incorrect. DAPF PC
    ;            EQUALS 3 H should be high. If this is wrong then the error
    ;            could be in the 8 to 1 MUX that outputs the memory request.
    ;            The output from that MUX should be low to initiate a memory
    ;            request. The logic from this point has been already been
    ;            tested by the MEM.REQ instruction.
    ;    Error 6:
    ;    Error 7:
    ;
    ;--

T.1:
        MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                         ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE
WAIT.T1.0:
        JMP [WAIT.T1.0]                  ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]          ;CLEAR COMPAT MODE AND SET IPL TO 1F
        MCOM LS[#FF] TO WR[0]
        MOV WR[0] TO LS[ERROR.MASK]      ;ERROR MASK = FFFFFF00
        CLR LS[PREVIOUS.ERROR]           ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[CPU] TO WR[0]             ;2 IN WR
        MOV WR[0] TO LS[MODULE.NUM]      ;SET UP MODULE CODE FOR CPU MODULE
        CLR LS[PC]                       ;INITIALIZE PC
        MOV LS[#8] TO WR[0]
        BIS LS[#80] TO WR[0]
        MOV WR[0] TO LS[T9]              ;T9 = 88 (END OF LOOP)
LOOP.T1:
        MEM.REQ[IB.FILL] ADRS[PC] DT[LONG]       ;FILL IB WITH DATA FROM MEMORY
        MOV MEM.DATA TO WR[1]            ;*WILL HANG HERE IF STUCK IN IB.FILL LOOP*
                                         ; *CHECK DATA RCVD H LOGIC*
LOOP.T1.1:
        JSR [ASSERT.NA]                  ;NO EXP AND REC DATA ON SKIP TEST
        MOV LS[#1] TO WR[0]
        MOV WR[0] TO LS[ERROR.NUMBER]    ;ERROR NUMBER = 1
        MOV LS[PC] TO WR[2]              ;STORE PC
        MOV IB.DATA TO OS                   ;PUT BYTE INTO OS & INC PC & SKIP IF IB.VALID
        SKIP                              ;ERROR IF HERE (NO SKIP ON IB VALID)
        JMP [T1.2]                        ;SKIP OK  GO TO NEXT PART
        MOV LS[ONES] TO WR[1]             ;SET ERRONEOUS DATA TO FLAG ERROR
        CLR WR[0]
        JSR [CHECK.RESULT]                ;REPORT ERROR
        JMP [T1.LOE]
T1.2:
        INC LS[ERROR.NUMBER]              ;INCREMENT ERROR NUMBER TO 2
        JSR [DEASSERT.NA]
        MOV LS[OS] TO WR[0]               ;GET WRITTEN DATA
        MOV WR[2] TO LS[T8]               ;T8 <- PC - 1
        MOV MEM.DATA TO WR[1]             ;* WILL HANG HERE IF STUCK IN CPU ISTREAM *
                                          ; * REQ LOOP RESULTING FROM IB.DATA TO OS *
                                          ;  * CHECK DATA RCVD H LOGIC OR *
                                          ;    * MEM REQ H LOGIC *
        MEM.REQ[READ.P] ADRS[T8] DT[BYTE]
        MOV MEM.DATA TO WR[1]             ;GET EXPECTED DATA
        JSR [CHECK.RESULT]                ;CHECK DATA
        JMP [T1.LOE]                      ;LOOP ON ERROR IF ENABLED
T1.3:
        INC LS[ERROR.NUMBER]              ;INCREMENT ERROR NUMBER TO 3
        MOV LS[PC] TO WR[0]
        MOV WR[2] TO WR[1]                ;GET ORIGINAL PC
        INC WR[1]                         ;EXPECTED = PC+1 (PC SHOULD INCREMENT)
        JSR [CHECK.RESULT]                ;CHECK PC
        JMP [T1.LOE]                      ;LOOP ON ERROR IF ENABLED
        INC WR[2]
        XOR LS[T9] WITH WR[2] TO Q,       ;COUNTER = 88(H)?
         DT(LONG)&SET.ALU.CC
        JMP.IF[NEQ] TO [LOOP.T1]          ;DO ANOTHER PATTERN IF NOT

T1.4.RES:
        MOV LS[T9] TO WR[0]               ;88 IN WR 0
        ADD LS[#4] TO WR[0]               ;8C IN WR 0
        MOV WR[0] TO LS[T8]               ;T8 = 8C
LOOP.T1.4:
        MOV LS[#4] TO WR[0]
        MOV WR[0] TO LS[ERROR.NUMBER]     ;SET ERROR NUMBER TO ERROR 4
        MOV LS[T9] TO WR[0]
        MOV WR[0] TO LS[PC]               ;PC = 88 (LONGWORD OF ZEROS IN MEMORY)
        MOV MEM.DATA TO WR[1]             ;THIS INSTRUCTION FREES MCT IN CASE
                                          ; HUNG ON DECODE REQUEST
        MEM.REQ[IB.FILL] ADRS[PC] DT[LONG]        ;FILL IB WITH ZEROES
T1.4:
        MOV IB.DATA TO OS                 ;PUT BYTE IN OS & INCREMENT PC &SKIP IF IB.VALID
        NOP
        MOV LS[OS] TO WR[0]               ;RECEIVED = OS
        CLR WR[1]                         ;EXPECTED = 0
        JSR [CHECK.RESULT]                ;CHECK OS FOR 0
        JMP [T1.LOE.4]
        MOV LS[PC] TO WR[2]
        XOR WR[2] WITH LS[T8] TO Q,
         DT(LONG)&SET.ALU.CC              ;FOUR BYTES LOOKED AT?
        JMP.IF[NEQ] TO [T1.4]             ;JUMP BACK IF NOT
T1.5:
        INC LS[ERROR.NUMBER]              ;INCREMENT ERROR NUMBER TO 5
        MOV IB.DATA TO OS                 ;GET NEXT BYTE (BYTE 0 OF NEXT LONGWORD)
        NOP
        MOV LS[OS] TO WR[0]
        MOV LS[#FF] TO WR[1]            ;EXPECTED = FF
        JSR [CHECK.RESULT]              ;CHECK OS FOR FF
        JMP [T1.LOE.4]

T1.6:
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 6
        MOV LS[INTERUPT.EN] TO WR[0]    ;SET UP MASK TO DISABLE MEM.REF
        BIS LS[DISABLE.MEM.REF] TO WR[0]
        MOV WR[0] TO LS[CONTROL.STATUS] ;DISABLE MEM.REF
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE

WAIT.T1.6.1:
        JMP [WAIT.T1.6.1]               ;WAIT FOR CONSOLE TO TAKE CONTROL

        MEM.REQ[READ.V.RCHK.IFILL] ADRS[PC] DT[LONG]
                                        ;THIS WILL CLEAR IB VALID

        MOV LS[INTERUPT.EN] TO WR[0]    ;SET UP TO ENABLE MEM.REF
        MOV WR[0] TO LS[CONTROL.STATUS] ;ENABLE MEM.REF
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE

WAIT.T1.6.2:
        JMP [WAIT.T1.6.2]               ;WAIT FOR CONSOLE TO TAKE CONTROL

LOOP.T1.6:
        SET.STATE.ZERO                  ;SPECIFY TO SKIP ON IB VALID
        CLR LS[PC]                      ;KEEP PC=3 CLEAR SO IB VALID STAYS CLEAR
        MOV IB.DATA TO OS               ;DECODE TO SKIP ON IB VALID
        JMP [T1.7]                      ;OK IF HERE - SHOULD NOT SKIP
        MOV LS[ONES] TO WR[1]           ;SET ERRONEOUS DATA TO FLAG ERROR
        CLR WR[0]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.T1.6]                 ;LOOP ON ERROR

T1.7:
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 7

LOOP.T1.7:
        CLR.STATE.ZERO                  ;DO NOT SKIP ON IB VALID
        CLR LS[PC]                      ;KEEP PC=3 CLEAR SO IB VALID STAYS CLEAR
        MOV IB.DATA TO OS               ;DECODE TO SKIP ON IB VALID
        JMP [END.T1]                    ;OK IF HERE - SHOULD NOT SKIP
        MOV LS[ONES] TO WR[1]           ;SET ERRONEOUS DATA TO FLAG ERROR
        CLR WR[0]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.T1.7]                 ;LOOP ON ERROR

T1.LOE:
        INC WR[2]                       ;INCREMENT BACK UP OF OLD PC
        BIT WR[2] WITH LS[#3(H)],
         DT(LONG)&SET.ALU.CC            ;ARE THE TWO LOWER BITS OF INCREMENTED
                                        ; PC CLEAR?
        DEC WR[2]                       ;RESORE BACKED UP OLD PC
        MOV WR[2] TO LS[PC]             ;IF WE ARE LOOPING ON ERROR THE PC MUST
                                        ; BE RESTORED SO THAT THE SAME BYTE
                                        ; WILL BE LOOKED AT
        JMP.IF[NEQ] TO [LOOP.T1.1]        ;LOOP ON ERROR W/O REWRITING IB
        JMP [LOOP.T1]                   ;LOOP ON ERROR REWRITING IB

T1.LOE.4:
        MOV LS[#3(H)] TO WR[3]          ;SET LOWER 2 BITS
        BIT WR[3] WITH LS[PC],
         DT(LONG)&SET.ALU.CC            ;ARE THE TWO LOWER BITS OF INCREMENTED
                                        ; PC CLEAR?
        DEC LS[PC]                      ;RESET PC
        JMP.IF[NEQ] TO [T1.4]           ;LOOP ON ERROR W/O REWRITING IB
        JMP [LOOP.T1.4]                 ;LOOP ON ERROR REWRITING IB
END.T1:
    .Page "TEST 2 - Compatability Mode Disables IB.REQ and Sign Extend Mem Tests"
    ;++
    ;
    ;
    ;
    ; Test Description:
    ;    This test checks that with compatability mode set IB.REQ is disabled.
    ; CSR 08 will be high to insure that it is indeed Compat Mode that is block-
    ; ing the Mem Request. Compat mode will be brought high and after passing
    ; through the CC Control Pal will force DAPH DATA REQ H low thus blocking
    ; IB prefetch from filling.
    ;    This test then checks the sign extension logic that could not be checked
    ; in ENKCB sign extension test because Memory had not yet been checked.
    ;
    ; Asumptions:
    ;    It is assumed that the "IB BUS AND LATCHES TEST" has been run
    ; successfully, and that all Memory diagnostics have run successfully.
    ;
    ; Test Steps:
    ;
    ; 1) Execute a mem.req to load IB prefetch with data.
    ; 2) Set Compat Mode.
    ; 3) Execute five decodes (MOV IB.DATA TO OS). On the fifth decode check
    ;    that the data is the same as the first decode and not what was in
    ;    the next memory location (IB should not have been rewritten).
    ; 4) Clear Compat Mode
    ; 5) Write and read a byte of zeros to memory, and check that it properly
    ;    sign extended.
    ; 6) Write and read a byte of '80' to memory, and check that it extended 1's.
    ; 7) Write and read a word of '80' to memory, and check that it extended 0's.
    ; 8) Write and read a word of '8000' to memory. Check for extended 1's.
    ;
    ; Errors:
    ;
    ;    Error 1 - Memory request not blocked by Compat Mode.
    ;    Error 2 - Byte of zeros not properly extended.
    ;    Error 3 - 1's not properly extended from byte.
    ;    Error 4 - Zeros not properly extended from word.
    ;    Error 5 - 1's not properly extended from word.
    ;
    ; Debug:
    ;
    ;    Error 1:
    ;    Error 2:
    ;
    ;
```

```
;
```

```
   ;--
T.2:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.T2.0:
        JMP [WAIT.T2.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        MCOM LS[#FF] TO WR[0]
        MOV WR[0] TO LS[ERROR.MASK]     ;ERROR MASK = FFFFFF00
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        MOV LS[#80] TO WR[0]
        BIS LS[#8] TO WR[0]
        BIS LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 8C
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE AND IPL TO 1F
LOOP.T2.1:
        MOV LS[#80] TO WR[0]
        BIS LS[#8] TO WR[0]
        MOV WR[0] TO LS[PC]             ;PC=88
        MEM.REQ[IB.FILL] ADRS[PC] DT[LONG]      ;FILL IB WITH ZERO'S
T2.1:
        MOV IB.DATA TO OS               ;PUT BYTE IN OS & INC PC & SKIP IF IB VALID
        NOP
        MOV LS[PC] TO WR[2]
        XOR LS[T8] WITH WR[2] TO Q,
         DT(LONG)&SET.ALU.CC            ;FOUR BYTES LOOKED AT?
        JMP.IF[NEQ] TO [T2.1]           ;GO BACK IF NOT
        MOV IB.DATA TO OS               ;GET BYTE & INC PC & SKIP IF IB VALID
        NOP
        CLR WR[1]                       ;EXPECTED = 0
        MOV LS[OS] TO WR[0]             ;GET OS
        JSR [CHECK.RESULT]              ;CHECK OS FOR 0
        JMP [LOOP.T2.1]                 ;LOOP ON ERROR IF ENABLED
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE


        CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 2
T2.2:
        CLR LS[T9]                      ;USED AS MAIN MEMORY PHYSICAL ADDRESS
        CLR WR[1]                       ;EXPECTED DATA IS ZEROS
        MEM.REQ[WRITE.P] ADRS[T9] DT[BYTE]
                                        ;SET UP TO WRITE MEMORY
        WRITE.MEM LS[ZERO]              ;WRITE ZERO TO MEMORY
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
                                        ;SET UP TO READ MEMORY
        MOV MEM.DATA TO WR[0]           ;GET DATA FROM MEMORY
        JSR [CHECK.RESULT]                  ;CHECK RESULT FOR ZEROS
        JMP [T2.2]                      ;LOOP ON ERROR IF ENABLED

        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 3
T2.3:
        CLR LS[T9]                      ;USED AS MAIN MEMORY PHYSICAL ADDRESS
        MOV LS[#FFFFFF00] TO WR[1]      ;EXPECTED DATA = FFFFFF80
        BIS LS[#80] TO WR[1]
        MEM.REQ[WRITE.P] ADRS[T9] DT[BYTE]
                                        ;SET UP TO WRITE MEMORY
        WRITE.MEM LS[#80]               ;WRITE 80 TO MEMORY
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
                                        ;SET UP TO READ MEMORY
        MOV MEM.DATA TO WR[0]           ;GET DATA FROM MEMORY
        JSR [CHECK.RESULT]              ;CHECK RESULT FOR ZEROS
        JMP [T2.3]                      ;LOOP ON ERROR IF ENABLED

        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 4
T2.4:
        CLR LS[T9]                      ;USED AS MAIN MEMORY PHYSICAL ADDRESS
        CLR WR[1]                       ;EXPECTED DATA IS 00000080
        BIS LS[#80] TO WR[1]
        MEM.REQ[WRITE.P] ADRS[T9] DT[WORD]
                                        ;SET UP TO WRITE MEMORY
        WRITE.MEM LS[#80]               ;WRITE 80 TO MEMORY
        MEM.REQ[READ.P] ADRS[T9] DT[WORD]
                                        ;SET UP TO READ MEMORY
        MOV MEM.DATA TO WR[0]           ;GET DATA FROM MEMORY
        JSR [CHECK.RESULT]              ;CHECK RESULT FOR ZEROS
        JMP [T2.4]                      ;LOOP ON ERROR IF ENABLED

        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 5
T2.5:
        CLR LS[T9]                      ;USED AS MAIN MEMORY PHYSICAL ADDRESS
        MOV LS[#FFFFFFFF] TO WR[1]      ;EXPECTED DATA IS FFFF8000
        BIC LS[#FFFF] TO WR[1]
        BIS LS[#8000] TO WR[1]
        MEM.REQ[WRITE.P] ADRS[T9] DT[WORD]
                                        ;SET UP TO WRITE MEMORY
        WRITE.MEM LS[#8000]             ;WRITE 8000 TO MEMORY
        MEM.REQ[READ.P] ADRS[T9] DT[WORD]
                                        ;SET UP TO READ MEMORY
        MOV MEM.DATA TO WR[0]           ;GET DATA FROM MEMORY
        JSR [CHECK.RESULT]              ;CHECK RESULT FOR ZEROS
        JMP [T2.5]                      ;LOOP ON ERROR IF ENABLED

END.T2:
    .Page "TEST 3 - Opcode Reg Test (M8390 CPU MODULE)"
    ;++
    ;
    ;
    ; Test Description:
    ;
    ;     This test checks to see that the opcode register, the data
    ;     paths to and from the opcode register and the control logic all work
    ;     correctly. Most of the data path used to check the opcode reg-
    ;     ister has already been tested in the "IB BUS AND LATCHES TEST".
    ;     The data path starts in memory where a memory request causes the
```

```
 ;      IB Prefetch register to be loaded with data. From there the data
```

```
;       is moved into the opcode register. The only way to tell whether
;       the data put in the opcode register is correct is to get the
;       data to the 2901 where it can be compared with the correct data.
;       So the data is moved into the OS register. From the OS register
;       the data is moved into the 2901 and checked. All eight bits of
;       the opcode register must be tested for stuck at one stuck at
;       zero. COMPAT MODE can enable either the opcode register or the
;       prefetch register. It has been tested for the prefetch register.
;       To test for the opcode register a WR must be set to all zeroes.
;       Then a MOV WR[] TO LS[] instruction is issued to bring COMPAT MODE
;       low. Then a decode instruction with CSR 08 clear should fill OS with
;       data from the opcode register. To distinguish whether the data
;       came from the prefetch register or the opcode register the prefetch
;       register and the opcode register must contain different data.
;
; Assumptions:
;
;       It's assumed that the "DECODE INSTRUCTION TEST" has been run and found
;       to be working. The 10 byte patterns that are used to test the opcode
;       register must be in memory.
;
; Test Steps:
;
;       1) Initialize LS[10] to the starting address of the data pattern in
;          main memory. Initialize COMPAT MODE  low by clearing a WR and
;          issuing a MOV WR[] TO LS[] instruction.
;       2) Load the IB Prefetch from memory with data to test the opcode
;          register.
;       3) Load a byte of data from IB Prefetch to the opcode register with
;          the instruction DECODE.OPC1. The jump control field should contain
;          a no jump skip on IB VALID.
;       4) To move the byte of data from the opcode register to the OS regis-
;          ter requires the writing of a new macro that looks like
;          DECODE,IFUNC/SPEC,IB.REQUEST/NOP,CM.HI/NOP,LOAD.OS/LOAD.OS,RDEST/
;          NOP,OPC.SPEC/SPEC
;       5) Load the byte of received data into WR[0].
;       6) Call the subroutine which generates the expected data.
;       7) Load the expected data into the WR[1].
;       8) If the data is correct and there's data left then go to step 2,
;          else if error go to error routine.
;       9) Load a WR with all ones then issue a MOV WR[] TO LS[] to set COMPAT
;          MODE.
;         10) Load the prefetch register and the opcode with different and known
;          data. If the data comes from the prefetch register then go to the
;          end of the test else go to error.
;
; Error:
;
;       Error 1 - The logic behind COMPAT MODE failed. (Data from prefetch)
;       Error 2 - The data from the opcode register is incorrect either as a
;                 result of the lines from the opcode register being stuck,
;                 the bits of the opcode latch being stuck, or as a result of
;                 faulty control logic.
;       Error 3 - A memory request was incorrectly initiated when IB.Request
;                 was clear.
;       Error 4 - PC was incremented when IB.Request was clear.
;
```

```
; Debug:
    ;
    ;     Error 1: If the data in the OS register is not what it should be then
    ;              it's probably from the prefetch register which means that
    ;              COMPAT MODE was low when it should have been high. The pos-
    ;              sible sources of error are the CM.RDEST PAL or the inputs to
    ;              that PAL, LOAD PSL and BUS Y D31.
    ;
    ;     Error 2: In analyzing the differences between the expected data and
    ;              the received data there are two catagories of error. The
    ;              first being an error in the latch or lines leading to it.
    ;              If it is this kind of error then received data will be con-
    ;              sistantly wrong in one bit, that bit being stuck either high
    ;              or low. The other kind of error has to do with the control
    ;              logic. Since there's no way of telling if the opcode register
    ;              was loaded, both decode instructions have an equal chance of
    ;              being at fault if no data came through. To clock the
    ;              data into the opcode register DAPB IRD STATE L is asserted. If
    ;              this line isn't asserted then check CM.RDEST PAL for possible
    ;              malfunction. To enable the opcode register out to the OS reg-
    ;              ister WCSH CSR 08 H has to be low as does DAPB COMPAT MODE H.
    ;              If WCSH CSR 08 H is incorrect then the line is bad. IF the
    ;              DAPB COMPAT MODE H is incorrect then it's possible that CM.
    ;              RDEST PAL is in error.
    ;     Error 3:  If IB.Request is clear then a memory request should not be
    ;              initiated. If a request is initiated then the most likely
    ;              source of error is the input WCSH CSR 08 H to IB.IR CONTROL
    ;              PAL. The input should be low (clear) causing the output,
    ;              DAPF PC EQUALS 3 H to be low which would prevent the initia-
    ;              tion of a memory request.
    ;
    ;     Error 4: See error 3
    ;--
T.3:
          MOV LS[BEGIN.TEST] TO WR[0]       ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
          MOV WR[0] TO LS[CONTROL.STATUS]   ;SET BIT15 IN CNTROL AND STATUS WORD
                                            ; BIT15 INDICATES START OF TEST TO CONSOLE
          MISC [SET.CP.ATTN]                ;ISSUE CPU ATTN TO CONSOLE
WAIT.T3.0:
          JMP [WAIT.T3.0]                   ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
          MOV LS[HI.IPL] TO WR[0]
          MOV WR[0] TO LS[PSL.HW]           ;INSURE THAT COMPAT MODE IS CLEAR
                                            ; SO THAT DATA IS SURE TO COME FROM THE
                                            ; OPCODE REGISTER.
          MCOM LS[#FF] TO WR[0]
          MOV WR[0] TO LS[ERROR.MASK]       ;ERROR MASK = FFFFFF00
          CLR LS[PREVIOUS.ERROR]            ;CLEAR PREVIOUS ERROR NUMBER
          MOV LS[#1] TO WR[0]               ;SET WR0 TO 1
          MOV WR[0] TO LS[ERROR.NUMBER]     ;SET ERROR NUMBER TO FIRST ERROR
          ROL WR[0]                         ;SET WR0 TO 2
          MOV WR[0] TO LS[MODULE.NUM]       ;SET UP MODULE CODE FOR CPU MODULE
          MOV LS[BIT3] TO WR[0]
          MOV WR[0] TO LS[PC]               ;INITIALIZE MEMORY POINTER TO FIRST OF
                                            ; MARCHING 1'S PATTERN.

          MOV LS[#80] TO WR[0]
          BIS LS[#8] TO WR[0]
          MOV WR[0] TO LS[T8]               ;T8 = 88 ;END OF MARCHING 1'S PATTERN
          DEC WR[0]
```

```
MOV WR[0] TO LS[T10]            ;T10 = 87
```

```
        INC WR[0]
        BIS LS[BIT2] TO WR[0]
        MOV WR[0] TO LS[T9]             ;T9 = 8C ;POINTER TO WORD OF 1'S
        CLR WR[3]
LOOP.T3.1:
        MEM.REQ[IB.FILL] ADRS[PC] DT[LONG]      ;FILL IB WITH MARCHING 1'S PATTERN
T3.1:
        MOV LS[#2] TO WR[0]
        MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR NUMBER = 2
        MOV LS[PC] TO WR[0]
        MOV WR[0] TO LS[T11]            ;SAVE PC
        DECODE.OPC1 ADRS[T3.JUST.IN.CASE],
         SKIP.IF(IB VALID)              ;MOVE IB TO OPCODE REGISTER & INC PC
T3.JUST.IN.CASE:
        NOP
        MEM.REQ[IB.FILL] ADRS[T9] DT[LONG]      ;SET NEW DATA IN IB
        TST WR[3],
         DT(LONG)&SET.ALU.CC            ;FIRST TIME THROUGH?
        JMP.IF[NEQ] TO [T3.NOT1]        ;ONLY DO TEST FOR ERROR 1 FIRST TIME THROUGH
        MOV LS[ONES] TO WR[3]           ;SET FLAG
        DEC LS[ERROR.NUMBER]            ;ERROR NUMBER = 1
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE
        MOV IB.DATA TO OS
        NOP
        MOV LS[OS] TO WR[0]             ;GET PREFETCH REG
        MOV LS[#FF] TO WR[1]            ;EXPECETED = FF
        JSR [CHECK.RESULT]
        JMP [T3.LOE]
        DEC LS[PC]                      ;RESTORE PC
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 2
T3.NOT1:
        MOV CM.IB.DATA TO OS            ;MOVE OPCODE REG TO OS, NOTE: DOES NOT INC PC!
        NOP
        MOV LS[OS] TO WR[0]             ;GET RECEIVED DATA
        MEM.REQ[READ.P] ADRS[T11] DT[BYTE]
        MOV MEM.DATA TO WR[1]           ;GET EXPECTED DATA
        JSR [CHECK.RESULT]              ;CHECK OPCODE REGISTER FOR CORRECT PATTERN
        JMP [T3.LOE]
        MOV LS[PC] TO WR[0]             ;GET UPDATED PC
        XOR LS[T9] WITH WR[0] TO Q,
         DT(LONG)&SET.ALU.CC            ;DONE?
        JMP.IF[NEQ] TO [LOOP.T3.1]              ;GO BACK IF NOT
T3.3:
        MOV LS[#3(H)] TO WR[0]
        MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR NUMBER = 3
        MOV LS[T9] TO WR[0]
        MOV WR[0] TO LS[PC]             ;PC = 8C
        MEM.REQ[IB.FILL] ADRS[PC] DT[LONG]      ;FILL IB WITH 1'S
        DECODE.OPC1 ADRS[T3.JUST.IN.CASE2],
         SKIP.IF(IB VALID)              ;MOVE IB TO OPCODE REGISTER & INC PC
T3.JUST.IN.CASE2:
        NOP
        MOV LS[T10] TO WR[0]
        MOV WR[0] TO LS[PC]                 ;PC = 87
        MOV CM.IB.DATA TO OS            ;MOVE DATA TO OS & SKIP IF IB.VALID
                                        ; W/O MEM REQUEST
        NOP
        MOV CM.IB.DATA TO OS            ;NOW - GET BYTE & SEE IF THERE WAS A
                                        ; MEM REQUEST
        NOP
        MOV LS[OS] TO WR[0]
        MOV LS[#FF] TO WR[1]
        JSR [CHECK.RESULT]              ;CHECK THAT NO MEMORY REQUEST WAS INITIATED
        JMP [T3.3]                      ;REDO ERROR
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 4
        MOV LS[PC] TO WR[0]             ;GET PC
        MOV LS[T10] TO WR[1]            ;EXPECTED = 87
        JSR [CHECK.RESULT]              ;CHECK THAT PC WAS NOT INCREMENTED
        JMP [T3.3]                      ;REDO ERROR
        JMP [END.T3]
T3.LOE:
        DEC LS[PC]                      ;IF WE ARE LOOPING ON ERROR THE PC MUST
                                        ; BE DECREMENTED SO THAT THE SAME BYTE
                                        ; WILL BE LOOKED AT
        MOV LS[BIT1] TO WR[2]
        INC WR[2]                       ;SET LOWER 2 BITS
        BIT WR[2] WITH LS[PC],
         DT(LONG)&SET.ALU.CC            ;ARE THE TWO LOWER BITS OF PC SET?
        JMP.IF[NEQ] TO [T3.1]           ;LOOP ON ERROR W/O REWRITING IB
        JMP [LOOP.T3.1]                 ;LOOP ON ERROR REWRITING IB
END.T3:
    .Page "TEST 4 - Spec Decode Test (M8390 CPU MODULE)"
    ;++
    ;
    ; Test Description:
    ;
    ;     This test checks the contents and the addressing of the SPEC DECODE
    ;     ROM. In accessing all of the locations in ROM, the IFUNC field of the
    ;     DECODE instruction along with the value of the IB BUS is used. A
    ;     change in the IFUNC field will cause a change in the type of DECODE
    ;     instruction being executed. There are three bits of IFUNC and there-
    ;     fore 8 different DECODE instructions that must be used to access all
    ;     of the ROM. A DECODE instruction which enables the SPEC DECODE ROM
    ;     causes a jump to a location in the WCS. The location jumped to in WCS
    ;     is a result of the lower eight bits from ROM and the upper bits from
    ;     the DECODE instruction. Since the upper bits will be the same there
    ;     are only 256 possible consecutive locations that can be jumped to and
    ;     each of these locations has an INC WR[3] instruction in it. Since the
    ;     block starts at zero the address jumped to can be calculated by
    ;     subtracting WR[3] from 256(which is the end of the block). The actual
    ;     address jumped to is loaded into WR[0]. The expected value will be
    ;     generated in a subroutine using loops if possible. The expected value
    ;     is loaded into WR[1] and the check result subroutine will be called.
    ;     If there's an error and looping on error is desired the counter which
    ;     points to the ROM location is not incremented so that the same address
    ;     is accessed. The upper address bit of the ROM is the NAND of the three
    ;     low bits of the IB BUS. To test each IB bit the patterns 0 through 7
    ;     must be used. The zero bit pattern has already been tested. The IFUNC
    ;     bits will be zero so a DECODE.SPEC instruction is required.
    ;
    ; Assumptions:
```

```
;
```

```
;       It is assumed that main memory contains the values needed to address
;       all the 512 bytes of ROM in SPEC DECODE. It is also assumed that the
;       OPCODE REGISTER TEST has been successfully run.
;
; Test Steps:
;
;       1) Initialize LS locations, memory locations , and WR's to be used.
;       2) Fill IB with data from memory that will force an incrementing
;          pattern into the SPEC DECODE ROM.
;       3) Issue a decode instruction with IFUNC = 0 that will JSR to an
;          address in the 256 block of INC WR[3]'s.
;       4) Subtract WR3 form 256 (decimal) and put his result in WR0
;       5) Put expected data in WR1 and call error subroutine.
;       6) Toggle bit 2 of pattern
;       7) Repeat steps 2-5
;       8) Add 8 to pattern & toggle bit 2
;       9) Repeat steps 2-5
;       10) Repeat steps 2-9 for IFUNC = 2 to 7
;       11) Issue a decode instruction with IFUNC = 0 and a floating 0
;           in the lower three bits of IB
;
; Error:
;
;       OTHER data : contents of IB0
;
;       Error 1 - Either the addressing into the ROM failed or the contents
;                 of the ROM are incorrect. IFUNC = 0
;       Error 2 - Either the addressing into the ROM failed or the contents
;                 of the ROM are incorrect. IFUNC = 1
;       Error 3 - Either the addressing into the ROM failed or the contents
;                 of the ROM are incorrect. IFUNC = 2
;       Error 4 - Either the addressing into the ROM failed or the contents
;                 of the ROM are incorrect. IFUNC = 3
;       Error 5 - Either the addressing into the ROM failed or the contents
;                 of the ROM are incorrect. IFUNC = 4
;       Error 6 - Either the addressing into the ROM failed or the contents
;                 of the ROM are incorrect. IFUNC = 5
;       Error 7 - Either the addressing into the ROM failed or the contents
;                 of the ROM are incorrect. IFUNC = 6
;       Error 8 - Either the addressing into the ROM failed or the contents
;                 of the ROM are incorrect. IFUNC = 7
;       Error 9 - Addressing into ROM failed. Input to NAND gate = 011
;       Error A - Addressing into ROM failed. Input to NAND gate = 101
;       Error B - Addressing into ROM failed. Input to NAND gate = 110
;
; Debug:
;
;       Error 1-8: There are three likely possibilities for error. The ROM could
;                have incorrect values in it, or the addressing could be wrong
;                which would cause dual addressing, there's an error in the
;                control logic which would cause there to be no next address
;                therefore causing a jump to some place in the block. The
;                control logic is be tested in a later test. The input to the
;                ROM can be determined by examining WR[2] and then subtracting
;                this value from 512.
;
;       Error 9-B: Either the NAND gate to the ROM is bad or suspect the lines
;               to the NAND.
```

```
   ;
   ;--
T.4:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.T4.0:
        JMP [WAIT.T4.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        MCOM LS[#FF] TO WR[0]
        MOV WR[0] TO LS[ERROR.MASK]     ;ERROR MASK = FFFFFF00
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        CLR LS[PC]                      ;ALWAYS LOOK AT LOW BYTE OF PREFETCH
        JSR [ASSERT.OTHER]
;
; WR2 - Data Pattern to put into IB (Memory location 94 will be used)
; WR3 - The block of 256 inc's will contain INC WR[3]'s
; T8  = 94 (Pointer to Memeory location 94)
; T9  - Pointer to answers (initially 98)
; T10 - Temporary storage
; T11 = 107 (End of loop)
; T12 = 0111 (B) (Bits to be toggled)
;
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 94
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T9]             ;T9 = 98
        MOV LS[#100] TO WR[0]
        ADD LS[#8] TO WR[0]
        DEC WR[0]
        MOV WR[0] TO LS[T11]            ;T11 = 107
        MOV LS[#8] TO WR[2]
        DEC WR[2]                       ;WR2 = 7
        MOV WR[2] TO LS[T10]
        MOV WR[2] TO LS[T12]            ;T12 = 0111 (B)
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;MMLOC94 = 7
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE TO FORCE DATA FROM
                                        ; IB PREFETCH RATHER THAN THE OPCODE REG
LOOP.T4.1:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA IN MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;    The 7 in DEC.ADRS + SPEC.DECODE ROM(IFUNC + IB<7:3> + NAND(IB<2:0>))
```

```
;
```

```
        OPC2/DECODE,DEC.ADRS/7,IFUNC/0,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H (256 DECIMAL) - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T4.1]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INCREMENT ANSWER POINTER
        XOR LS[T12] TO WR[2]                    ;TOGGLE BITS 0-2 OF WR2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
LOOP2.T4.1:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/0,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;GET RECEIVED
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP2.T4.1]                        ;LOOP ON ERROR IF ENABLED
        INC LS[T9]
        ADD LS[#8] TO WR[2]                     ;ADD 8H TO PATTERN &
        XOR LS[T12] TO WR[2]                    ; TOGGLE BITS 0-2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
        MOV LS[T11] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 107 ?
        JMP.IF[NEQ] TO [LOOP.T4.1]              ;KEEP LOOPIN IF NOT
T4.2:
        INC LS[ERROR.NUMBER]                    ;ERROR NUMBER = 2
        MOV LS[#8] TO WR[2]
        DEC WR[2]                               ;RESET PATTERN TO 7
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;RESET PATTERN TO 7
LOOP.T4.2:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA IN MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;    The 7 in DEC.ADRS + SPEC.DECODE ROM(IFUNC + IB<7:3> + NAND(IB<2:0>))
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H (256 DECIMAL) - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T4.2]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INCREMENT ANSWER POINTER
        XOR LS[T12] TO WR[2]                    ;TOGGLE BITS 0-2 OF WR2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
LOOP2.T4.2:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;GET RECEIVED
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP2.T4.2]                        ;LOOP ON ERROR IF ENABLED
        INC LS[T9]
        ADD LS[#8] TO WR[2]                     ;ADD 8H TO PATTERN &
        XOR LS[T12] TO WR[2]                    ; TOGGLE BITS 0-2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
        MOV LS[T11] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 107 ?
        JMP.IF[NEQ] TO [LOOP.T4.2]              ;KEEP LOOPIN IF NOT
T4.3:
        INC LS[ERROR.NUMBER]                    ;ERROR NUMBER = 3
        MOV LS[#8] TO WR[2]
        DEC WR[2]                               ;RESET PATTERN TO 7
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;RESET PATTERN TO 7
LOOP.T4.3:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA IN MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;    The 7 in DEC.ADRS + SPEC.DECODE ROM(IFUNC + IB<7:3> + NAND(IB<2:0>))
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/2,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H (256 DECIMAL) - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T4.3]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INCREMENT ANSWER POINTER
        XOR LS[T12] TO WR[2]                    ;TOGGLE BITS 0-2 OF WR2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
LOOP2.T4.3:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/2,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;GET RECEIVED
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP2.T4.3]                        ;LOOP ON ERROR IF ENABLED
        INC LS[T9]
        ADD LS[#8] TO WR[2]                     ;ADD 8H TO PATTERN &
        XOR LS[T12] TO WR[2]                    ; TOGGLE BITS 0-2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
        MOV LS[T11] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 107 ?
        JMP.IF[NEQ] TO [LOOP.T4.3]              ;KEEP LOOPIN IF NOT
T4.4:
        INC LS[ERROR.NUMBER]                    ;ERROR NUMBER = 4
        MOV LS[#8] TO WR[2]
        DEC WR[2]                               ;RESET PATTERN TO 7
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;RESET PATTERN TO 7
LOOP.T4.4:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA IN MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;     The 7 in DEC.ADRS + SPEC.DECODE ROM(IFUNC + IB<7:3> + NAND(IB<2:0>))
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/3,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H (256 DECIMAL) - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T4.4]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INCREMENT ANSWER POINTER
        XOR LS[T12] TO WR[2]                    ;TOGGLE BITS 0-2 OF WR2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
LOOP2.T4.4:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/3,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;GET RECEIVED
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP2.T4.4]                        ;LOOP ON ERROR IF ENABLED
        INC LS[T9]
        ADD LS[#8] TO WR[2]                     ;ADD 8H TO PATTERN &
        XOR LS[T12] TO WR[2]                    ; TOGGLE BITS 0-2
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
        MOV LS[T11] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 107 ?
        JMP.IF[NEQ] TO [LOOP.T4.4]              ;KEEP LOOPIN IF NOT
T4.5:
        INC LS[ERROR.NUMBER]                    ;ERROR NUMBER = 5
        MOV LS[#8] TO WR[2]
        DEC WR[2]                               ;RESET PATTERN TO 7
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;RESET PATTERN TO 7
LOOP.T4.5:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA IN MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;    The 7 in DEC.ADRS + SPEC.DECODE ROM(IFUNC + IB<7:3> + NAND(IB<2:0>))
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/4,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H (256 DECIMAL) - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T4.5]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INCREMENT ANSWER POINTER
        XOR LS[T12] TO WR[2]                    ;TOGGLE BITS 0-2 OF WR2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
LOOP2.T4.5:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/4,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;GET RECEIVED
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP2.T4.5]                        ;LOOP ON ERROR IF ENABLED
        INC LS[T9]
        ADD LS[#8] TO WR[2]                     ;ADD 8H TO PATTERN &
        XOR LS[T12] TO WR[2]                    ; TOGGLE BITS 0-2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
        MOV LS[T11] TO Q
        CMP WR[2] WITH Q,
        DT(LONG)&SET.ALU.CC                         ;PATTERN = 107 ?
        JMP.IF[NEQ] TO [LOOP.T4.5]                 ;KEEP LOOPIN IF NOT
T4.6:
        INC LS[ERROR.NUMBER]                       ;ERROR NUMBER = 6
        MOV LS[#8] TO WR[2]
        DEC WR[2]                                  ;RESET PATTERN TO 7
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                          ;RESET PATTERN TO 7
LOOP.T4.6:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]         ;FILL IB WITH DATA IN MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]              ;SET "OTHER" DATA
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;    The 7 in DEC.ADRS + SPEC.DECODE ROM(IFUNC + IB<7:3> + NAND(IB<2:0>))
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/5,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                             ;RECEIVED = 100H (256 DECIMAL) - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                      ;GET EXPECTED FROM MEMORY
        JSR [CHECK.RESULT]                         ;CHECK
        JMP [LOOP.T4.6]                            ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                                 ;INCREMENT ANSWER POINTER
        XOR LS[T12] TO WR[2]                       ;TOGGLE BITS 0-2 OF WR2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                          ;WRITE NEW PATTERN TO MMLOC94
LOOP2.T4.6:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]         ;FILL IB WITH DATA FROM MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]              ;SET "OTHER" DATA
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/5,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                             ;GET RECEIVED
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                      ;GET EXPECTED
        JSR [CHECK.RESULT]                         ;CHECK
        JMP [LOOP2.T4.6]                           ;LOOP ON ERROR IF ENABLED
        INC LS[T9]
        ADD LS[#8] TO WR[2]                        ;ADD 8H TO PATTERN &
        XOR LS[T12] TO WR[2]                       ; TOGGLE BITS 0-2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                          ;WRITE NEW PATTERN TO MMLOC94
        MOV LS[T11] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                       ;PATTERN = 107 ?
        JMP.IF[NEQ] TO [LOOP.T4.6]                 ;KEEP LOOPIN IF NOT
T4.7:
        INC LS[ERROR.NUMBER]                       ;ERROR NUMBER = 7
        MOV LS[#8] TO WR[2]
        DEC WR[2]                                  ;RESET PATTERN TO 7
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;RESET PATTERN TO 7
```

```
LOOP.T4.7:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA IN MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;    The 7 in DEC.ADRS + SPEC.DECODE ROM(IFUNC + IB<7:3> + NAND(IB<2:0>))
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/6,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H (256 DECIMAL) - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T4.7]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INCREMENT ANSWER POINTER
        XOR LS[T12] TO WR[2]                    ;TOGGLE BITS 0-2 OF WR2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
LOOP2.T4.7:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/6,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;GET RECEIVED
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP2.T4.7]                        ;LOOP ON ERROR IF ENABLED
        INC LS[T9]
        ADD LS[#8] TO WR[2]                     ;ADD 8H TO PATTERN &
        XOR LS[T12] TO WR[2]                    ; TOGGLE BITS 0-2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
        MOV LS[T11] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 107 ?
        JMP.IF[NEQ] TO [LOOP.T4.7]              ;KEEP LOOPIN IF NOT
T4.8:
        INC LS[ERROR.NUMBER]                    ;ERROR NUMBER = 8
        MOV LS[#8] TO WR[2]
        DEC WR[2]                               ;RESET PATTERN TO 7
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;RESET PATTERN TO 7
LOOP.T4.8:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA IN MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
```

```
;	The 7 in DEC.ADRS + SPEC.DECODE ROM(IFUNC + IB<7:3> + NAND(IB<2:0>))
```

```
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/7,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H (256 DECIMAL) - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T4.8]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INCREMENT ANSWER POINTER
        XOR LS[T12] TO WR[2]                    ;TOGGLE BITS 0-2 OF WR2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
LOOP2.T4.8:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC94
        MOV LS[T10] TO WR[3]
        MOV WR[3] TO LS[ADDRESS.DATA]           ;SET "OTHER" DATA
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/7,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;GET RECEIVED
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTED
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP2.T4.8]                        ;LOOP ON ERROR IF ENABLED
        INC LS[T9]
        ADD LS[#8] TO WR[2]                     ;ADD 8H TO PATTERN &
        XOR LS[T12] TO WR[2]                    ; TOGGLE BITS 0-2
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE NEW PATTERN TO MMLOC94
        MOV LS[T11] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 107 ?
        JMP.IF[NEQ] TO [LOOP.T4.8]              ;KEEP LOOPIN IF NOT
        JSR [DEASSERT.OTHER]
        INC LS[ERROR.NUMBER]                    ;ERROR NUMBER = 9
        MOV LS[#40] TO WR[2]
        ADD LS[#8] TO WR[2]
        ADD LS[#2] TO WR[2]
        ADD LS[#1] TO WR[2]                     ;WR2 = 4B
LOOP.T4.9:
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;MMLOC94 = 4B
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;IB = 4B
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/0,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]
        MOV LS[#10] TO WR[1]
        ADD LS[#2] TO WR[1]
        ADD LS[#1] TO WR[1]                     ;WR1 = 13
        JSR [CHECK.RESULT]                      ;CHECK
        JSR [LOOP.T4.9]                         ;LOOP ON ERROR IF ENABLED
        INC LS[ERROR.NUMBER]                    ;ERROR NUMBER = A
        ADD LS[#2] TO WR[2]                     ;WR2 = 4D
```

```
LOOP.T4.10:
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;MMLOC94 = 4D
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;IB = 4D
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/0,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]
        MOV LS[#10] TO WR[1]
        ADD LS[#2] TO WR[1]
        ADD LS[#1] TO WR[1]                     ;WR1 = 13
        JSR [CHECK.RESULT]                      ;CHECK
        JSR [LOOP.T4.10]                        ;LOOP ON ERROR IF ENABLED
        INC LS[ERROR.NUMBER]                    ;ERROR NUMBER = B
        ADD LS[#1] TO WR[2]                     ;WR2 = 4E
LOOP.T4.11:
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;MMLOC94 = 4E
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;IB = 4E
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/0,BPC/NOP,OPC.SPEC/SPEC,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]
        MOV LS[#10] TO WR[1]
        ADD LS[#2] TO WR[1]
        ADD LS[#1] TO WR[1]                     ;WR1 = 13
        JSR [CHECK.RESULT]                      ;CHECK
        JSR [LOOP.T4.11]                        ;LOOP ON ERROR IF ENABLED
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]                 ;CLEAR COMPAT MODE
END.T4:
   .Page "TEST 5 - Opcode Decode Test (M8390 CPU MODULE)"
   ;++
   ;
   ;
   ; Test Description:
   ;
   ;    This test checks to see that the addressing and the contents of the
   ;    OPCODE DECODE ROM are correct. As with the SPEC DECODE ROM the IFUNC
   ;    bits are part of the addressing lines, in this case however only CSR
   ;    bits 9 and 10 are inputs which means that only four DECODE instruc-
   ;    tions are required to access all of the ROM. The outputs of the ROM
   ;    are micro addresses into the WCS. To determine if the DECODE instruc-
   ;    tion has jumped to the right micro address there is a block of 256 INC
   ;    WR[3] instructions each one corresponding to one of the 256 possible
   ;    micro addresses that can be jumped to. At the end of the block there
   ;    is a return. WR[3] will contain 256 minus the number of the address
   ;    jumped to. To determine if the correct address was jumped to subtract
   ;    WR[3] from 256 and load it into WR[0]. Generate the expected micro
   ;    address and load it into WR[1] and call CHECK RESULT. If ther's an
   ;    Error  and loop on error then don't increment WR[2] which causes the
   ;    same location in the ROM to be looked at which means that the same
   ;    micro address is jumped to.
   ;
   ; Assumptions:
   ;
```

```
  ;     It is assumed that the data to check each ROM location is in memory
```

```
        ;       and that it can be loaded onto the IB BUS.
        ;
        ; Test Steps:
        ;
        ;       1) Initialize WR's, LS locations, and MM locations that are to be used.
        ;       2) Move pattern into IB
        ;       3) Execute a DECODE instruction with IFUNC = 0 that will JSR to a
        ;          block of 256 INC WR[3]'s
        ;       4) Subtract wr3 from 100H and put into wr0
        ;       5) Get expected data from memory & call error check routine
        ;       6) Inc pattern
        ;       7) Repeat steps 2-6 until Pattern =100
        ;       8) Reinitialize Pattern
        ;       9) Repeat steps 2-8 for IFUNC = 2,3, and 4
        ;
        ; Error:
        ;
        ;       Error 1 - Either the addressing logic or the contents of the OPCODE
        ;                 DECODE ROM is incorrect.
        ;
        ; Debug:
        ;
        ;       Error 1: There are three general possibilities for an error: the
        ;                addressing to the ROM, the contents of the ROM or the control
        ;                logic for the ROM, is bad. In probably all cases the jump from
        ;                the DECODE will cause you to go to some place in the block of
        ;                INC WR[3]. If there is an error in addressing you will tend to
        ;                jump to the same micro address depending upon what bits are
        ;                stuck in what positions. If the control logic is in error, any
        ;                where in the block is just as likely as anywhere else, but the
        ;                control logic will be testsd later. The contents of the ROM
        ;                being in error are just as likely in one place as another. The
        ;                input to the ROM can be determined by subtracting WR[2] from
        ;                512.
        ;
        ;--
T.5:
        MOV LS[BEGIN.TEST] TO WR[0]       ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                          ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]                ;ISSUE CPU ATTN TO CONSOLE
WAIT.T5.0:
        JMP [WAIT.T5.0]                   ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]           ;CLEAR COMPAT MODE
        MCOM LS[#FF] TO WR[0]
        MOV WR[0] TO LS[ERROR.MASK]       ;ERROR MASK = FFFFFF00
        CLR LS[PREVIOUS.ERROR]            ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]               ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]     ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                         ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]       ;SET UP MODULE CODE FOR CPU MODULE
        MOV LS[INTERUPT.EN] TO WR[0]
        MOV WR[0] TO LS[CONTROL.STATUS] ;INSURE ALL INTERRUPTS ARE ABSENT
        MISC [SET.CP.ATTN]                ;ISSUE CPU ATTN TO CONSOLE
WAIT.T5.1:
        JMP [WAIT.T5.1]                   ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
```

```
;
```

```
; T8  = 94 (Pointer to memory locaton 94)
; T9  = Pointer to answers, initially 298
; T10 = General storage
; WR2 = Pattern to put into IB
; T11 = Answer storage for first half
; T12 = Answer count for first half
;
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 94
        ADD LS[#200] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T9]             ;T9 = 298
        CLR WR[2]                       ;INITIAL PATTERN = 0
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE INITIAL PATTERN IN MMLOC 94
        CLR LS[PC]
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE SO THAT DATA IS FORCED
                                        ; FROM PREFETCH & SET IPL TO 1F
;
;Since the first half of the ROM is repetitive. The expected data will use a
;count/answer method. This initialixes the first answer & count
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T12]         ;T12 = INITIAL COUNT
        INC LS[T9]                      ;INCREMENT MEMORY POINTER
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T11]         ;T11 = FIRST ANSWER
LOOP.T5.1:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC 94
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;       The 7 in DEC.ADRS + OPCODE.DECODE ROM(The lower two bits of IFUNC + IB)
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/0,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H - WR3
        MOV LS[T11] TO WR[1]                    ;GET EXPECTED
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T5.1]                         ;LOOP ON ERROR IF ENABLED
        DEC LS[T12]                             ;DECREMENT COUNT
        MOV LS[T12] TO WR[0]
        TST WR[0],
         DT(LONG)&SET.ALU.CC                    ;DOWN TO ZERO ?
        JMP.IF[NEQ] TO [T5.1.SAME]              ;JUMP IF NOT
        INC LS[T9]
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T12]                 ;GET NEXT COUNT
        INC LS[T9]
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T11]                 ;GET NEXT ANSWER
T5.1.SAME:
        INC WR[2]                               ;INC PATTERN
```

```
MOV WR[2] TO LS[T10]
```

```
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE PATTERN IN MMLOC 94
        MOV LS[#100] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 100 ?
        JMP.IF[NEQ] TO [LOOP.T5.1]              ;LOOP BACK IF NOT
T5.2:
        CLR WR[2]                       ;INITIAL PATTERN = 0
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE INITIAL PATTERN IN MMLOC 94
LOOP.T5.2:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC 94
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;       The 7 in DEC.ADRS + OPCODE.DECODE ROM(The lower two bits of IFUNC + IB)
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H - WR3
        MOV LS[T11] TO WR[1]
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T5.2]                         ;LOOP ON ERROR IF ENABLED
        DEC LS[T12]                             ;DECREMENT COUNT
        MOV LS[T12] TO WR[0]
        TST WR[0],
         DT(LONG)&SET.ALU.CC                    ;DOWN TO ZERO ?
        JMP.IF[NEQ] TO [T5.2.SAME]              ;JUMP IF NOT
        INC LS[T9]
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T12]                 ;GET NEXT COUNT
        INC LS[T9]
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T11]                 ;GET NEXT ANSWER
T5.2.SAME:
        INC WR[2]                               ;INC PATTERN
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE PATTERN IN MMLOC 94
        MOV LS[#100] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 100 ?
        JMP.IF[NEQ] TO [LOOP.T5.2]              ;LOOP BACK IF NOT
T5.3:
        CLR WR[2]                       ;INITIAL PATTERN = 0
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE INITIAL PATTERN IN MMLOC 94
        MOV LS[#00F0] TO WR[0]
        ADD LS[#200] TO WR[0]
        ADD LS[#8] TO WR[0]
        ADD LS[#2] TO WR[0]
        MOV WR[0] TO LS[T9]            ;T9 = 2FA
LOOP.T5.3:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC 94
        CLR WR[3]
```

```
;The following macro JSR's to the location produced by:
```

```
;
;       The 7 in DEC.ADRS + OPCODE.DECODE ROM(The lower two bits of IFUNC + IB)
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/2,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTE FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T5.3]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INC ANSWER POINTER
        INC WR[2]                               ;INC PATTERN
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE PATTERN IN MMLOC 94
        MOV LS[#100] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 100 ?
        JMP.IF[NEQ] TO [LOOP.T5.3]              ;LOOP BACK IF NOT
T5.4:
        CLR WR[2]                       ;INITIAL PATTERN = 0
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE INITIAL PATTERN IN MMLOC 94
LOOP.T5.4:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH DATA FROM MMLOC 94
        CLR WR[3]
;The following macro JSR's to the location produced by:
;
;       The 7 in DEC.ADRS + OPCODE.DECODE ROM(The lower two bits of IFUNC + IB)
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/3,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                          ;RECEIVED = 100H - WR3
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO WR[1]                   ;GET EXPECTE FROM MEMORY
        JSR [CHECK.RESULT]                      ;CHECK
        JMP [LOOP.T5.4]                         ;LOOP ON ERROR IF ENABLED
        INC LS[T9]                              ;INC ANSWER POINTER
        INC WR[2]                               ;INC PATTERN
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                       ;WRITE PATTERN IN MMLOC 94
        MOV LS[#100] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC                    ;PATTERN = 100 ?
        JMP.IF[NEQ] TO [LOOP.T5.4]              ;LOOP BACK IF NOT
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]                 ;CLEAR COMPAT MODE
END.T5:
  .Page "TEST 6 - Decode Control Logic Test (M8390 CPU MODULE)"
   ;++
   ;
   ;
   ; Test Description:
   ;
   ;    This test checks the control logic of the SPEC DECODE ROM and
;       the OPCODE DECODE ROM. The control logic is designed in such a way
;       that if one ROM is enabled the other ROM is not. To enable either of
;       the ROMs the instruction must be a DECODE and the JCTL field
;       must be valid. Besides the previously mentioned constraints, there
;       are five other signals that play a part in whether the ROMs are
;       enabled. These signals are the OPC/SPEC bit(CSR 04), CSR bits 9 and 10
;       (IFUNC), INTERRUPT REQUEST, and MASK. To determine which ROM has been
;       selected the eight bits from the IB BUS will be all zero's and the
;       IFUNC bits will all be ones when not otherwise stated. A DECODE
;       instruction which enables either the OPCODE DECODE ROM or the SPEC
;       DECODE ROM causes a jump to a location in WCS. The location jumped to
;       in WCS is a result of the lower eight bits from the ROM and the upper
;       bits from the DECODE instruction. Since the upper bits will be the
;       same there are only 256 consecutive possible locations that can be
;       jumped to and each of these locations has a INC WR[3] in it. If the
;       SPEC DECODE ROM is selected then the DECODE instruction will cause a
;       jump to the INC WR[3] instruction selected by the appropriate SPEC
;       DECODE ROM location. If the OPCODE DECODE ROM is selected then there
;       will be a jump to the INC WR[3] instruction  selected by the approp-
;       riate OPCODE DECODE ROM location. Since the location jumped to is
;       known (from data in IB, the DEC.ADRS  and the IFUNC Bits) then it is
;       also known how many times WR[3] has been incremented before it reached
;       the end of the block. By comparing the value of WR[3] to the expected
;       value we are able to determine if the location jumped to was correct.
;       To test all of these five signals for stuck at one and stuck at zero
;       there are 5 conditions required.
;
;       Condition # CSR4    IFUNC<0> IFUNC<1> INT_REQ  MASK  ROM Selected
;
;            1         1        1        1        1       1   Spec Decode
;            2         1        1        1        1       0   Opcode Decode
;            3         1        0        1        1       1   Opcode Decode
;            4         1        1        0        1       1   Opcode Decode
;            5         1        1        1        0       1   Opcode Decode
;
;       Note: CSR4 was tested in the previous two tests and will be set
;              throughout this test.
;
;   Assumptions:
;
;       It is assumed that memory is set up so that the IB BUS is loaded with
;       zero's for the duration of this test. It's also assumed that the tests
;       for the SPEC DECODE ROM and the OPCODE DECODE ROM have been run
;       successfully.
;
;   Test Steps:
;
;       1) Issue a decode instruction with conditions set as in condition #1
;          as described above
;       2) Subtrat wr3 from 256 (100H) and generate expected data
;       3) Call check routine
;       4) Repeat steps 1-3 for condition #'s 2 through 5
;
;
;   Error:
;
;       Error 1 - The SPEC DECODE ROM was not enabled with conditions set as
;                 described above in Condition #1
```

```
;	Error 2 - The OPCODE DECODE ROM was not enabled with conditions set as
```

```
    ;               described above in Condition #2
    ;     Error 3 - The OPCODE DECODE ROM was not enabled with conditions set as
    ;               described above in Condition #3
    ;     Error 4 - The OPCODE DECODE ROM was not enabled with conditions set as
    ;               described above in Condition #4
    ;     Error 5 - The OPCODE DECODE ROM was not enabled with conditions set as
    ;               described above in Condition #5
    ;
    ; Debug:
    ;
    ;     Error 1: The four input negated OR gate did not produce a low to
    ;              the two input negated OR and the enable of OPCODE DECODE
    ;
    ;              -OR- The two input negated OR did not produce a high to
    ;                   the enble of SPEC DECODE
    ;
    ;     Error 2: The four input negated OR gata did not produce a high
    ;                 with MASK INTS low and the odher inputs high.
    ;
    ;     Error 3: The four input negated OR gate -kd not produce a high
    ;                 with WSCH CSR 09 low and the other inputs high.
    ;
    ;     Error 4: The four input negated OR gata did not produce a high
    ;                 with WSCH CSR 10 low and the othar inputs high.
    ;
    ;     Error 5: The four input negated OR gate did not produce a high
    ;                 with INTERR REQ low and the othar inputs high.
    ;
    ;
    ;--
T.6:
        MOV LS[BEGIN.TEST] TO WR[0]       ;SDT BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                          ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]                ;ISSUE CPU ATTN TO CONSOLE
WAIT.T6.0:
        JMP [WAIT.T6.0]                   ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MCOM LS[#FF] TO WR[0]
        MOV WR[0] TO LS[ERROR.MASK]       ;ERROR MASK = FFFFFF00
        CLR LS[PREVIOUS.ERROR]            ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]           ;SET IPL TO 1F & CLEAR COMPAT MODE
        MOV LS[#1] TO WR[0]               ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]     ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                         ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]       ;SET UP MODULE CODE FOR CPU MODULE
        MOV LS[CONSOLE.HALT] TO WR[0]
        BIS LS[INTERUPT.EN] TO WR[0]
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BITS TO BRING UP CONSOLE HALT INTERRUPT
                                          ; THUS MAKING INTERR REQ HIGH
        MISC [SET.CP.ATTN]                ;CALL CONSOLE
WAIT.T6.1:
        JMP [WAIT.T6.1]                   ;WAIT FOR CONSOLE
        MOV LS[#80] TO WR[0]
        ADD LS[#8] TO WR[0]
        MOV WR[0] TO LS[T8]               ;T8 = 88 (POINTER TO WORD OF 0'S)
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]        ;FILL IB WITH 0'S
        MOV LS[HI.IPL] TO WR[0]
```

```
BIS LS[BIT31] TO WR[0]
```

```
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE SO THAT DATA IS FORCED
                                        ; FROM PREFETCH & SET IPL TO 1F
LOOP.T6.1:
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/3,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                  ;RECEIVED = 100H - WR3
        MOV LS[#000F] TO WR[1]
        MOV LS[#00F0] TO WR[2]
        ADD WR[2] TO WR[1]              ;EXPECTED = FF (LOCATION C1 OF SPEC DECODE)
        JSR [CHECK.RESULT]              ;CHECK THAT SPEC DECODE WAS ENABLED
        JMP [LOOP.T6.1]                 ;LOOP ON ERROR IF ENABLED
T6.2:
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 2
LOOP.T6.2:
        CLR WR[3]
        MISC2 [MASK.INTERRUPTS]         ;SET MASK INTS LOW FOR ONE CYCLE
        MISC2 [MASK.INTERRUPTS]         ;V1.1 Repeat above to get valid data.
        OPC2/DECODE,DEC.ADRS/7,IFUNC/3,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                  ;RECEIVED = 100H - WR3
        MOV LS[#00F0] TO WR[1]
        BIC LS[BIT6] TO WR[1]
        ADD LS[#4] TO WR[1]
        ADD LS[#8] TO WR[1]             ;EXPECTED = BC (LOCATION 300 OF OPCODE DECODE)
        JSR [CHECK.RESULT]              ;CHECK THAT OPCODE DECODE WAS ENABLED
        JMP [LOOP.T6.2]
T6.3:
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 3
LOOP.T6.3:
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/2,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                  ;RECEIVED = 100H - WR3
        MOV LS[#00F0] TO WR[1]
        BIC LS[BIT5] TO WR[1]
        ADD LS[#8] TO WR[1]             ;EXPECTED = D8 (LOCATION 200 OF OPCODE DECODE)
        JSR [CHECK.RESULT]              ;CHECK THAT OPCODE DECODE WAS SELECTED
        JMP [LOOP.T6.3]                 ;LOOP ON ERROR IF ENABLED
T6.4:
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 4
LOOP.T6.4:
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                  ;RECEIVED = 100H - WR3
        MOV LS[#000F] TO WR[1]
        BIC LS[BIT1] TO WR[1]
        ADD LS[#80] TO WR[1]            ;EXPECTED = 8D (LOCATION 100 OF OPCODE DECODE)
        JSR [CHECK.RESULT]              ;CHECK THAT OPCODE DECODE WAS SELECTED
        JMP [LOOP.T6.4]                 ;LOOP ON ERROR IF ENABLED
T6.5:
        INC LS[ERROR.NUMBER]            ;INCREMENT ERROR NUMBER TO 5
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE SO INTERRUPT CAN BE LOWERED
        MOV LS[BIT16] TO WR[0]
        MOV WR[0] TO LS[CONTROL.STATUS] ;TO CLEAR INTERRUPTS
```

```
	MISC [SET.CP.ATTN]		;CALL CONSOLE (HEY! CONSOLE!)
```

```
WAIT.T6.5:
        JMP [WAIT.T6.5]                 ;WAIT FOR CONSOLE
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;RESET COMPAT MODE
LOOP.T6.5:
        CLR WR[3]
        OPC2/DECODE,DEC.ADRS/7,IFUNC/3,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/JSR
        SUB WR[3] FROM LS[#100] TO Q
        MOV Q TO WR[0]                  ;RECEIVED = 100H - WR3
        MOV LS[#00F0] TO WR[1]
        BIC LS[BIT6] TO WR[1]
        ADD LS[#4] TO WR[1]
        ADD LS[#8] TO WR[1]             ;EXPECTED = BC (LOCATION 300 OF OPCODE DECODE)
        JSR [CHECK.RESULT]              ;CHECK THAT OPCODE DECODE WAS SELECTED
        JMP [LOOP.T6.5]                 ;LOOP ON ERROR IF ENABLED
T6.CLEANUP:
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE AND RESET IPL TO 1F
END.T6:
  .Page "TEST 7 - OPC2 ROM Test (M8390 CPU MODULE)"
   ;++
   ;
   ;
   ; Test Description:
   ;
   ;    This test checks to see that the contents and the addressing of the
   ;    OPC2 ROM is correct. The inputs to the OPC2 ROM are the eight IB BUS
   ;    lines and COMPAT MODE. The IB BUS lines can be manipulated with DECODE
   ;    instructions and COMPAT MODE will be set low for half of the test
   ;    and high for the other half of the test. In testing OPC2 ROM it is
   ;    neccessary to test other signals and PALS. The outputs of the ROM are
   ;    DT CLASS 0, DT CLASS 1, CC CLASS 0, and CC CLASS 1. To test CC CLASS 0
   ;    and CC CLASS 1 they must be followed through the BRANCH COND CC CLASS
   ;    PAL and out as OPC TYPE 1 and OPC TYPE 0. OPC TYPE 0 and OPC TYPE 1
   ;    are inputs to the PSL CC PAL.The outputs of the PSL CC PAL can be read
   ;    on the D BUS. The other outputs of the OPC2 ROM can be traced to the
   ;    DATA TYPE CTL PAL which are output onto the D BUS and the SIZE REG.
   ;    Once the outputs of the OPC2 ROM are set up by varying IB and COMPAT
   ;    MODE then a MOV instruction is issued from LS [7E](with 2 in the CC
   ;    field) to enable the contents of the size reg onto bits 0 and 1 of the
   ;    D BUS for checking. The contents of bits 0 and 1 should be the same as
   ;    DT CLASS 0 and DT CLASS 1. The subroutine Check Result is called to
   ;    compare the expected value and the actual value and foop on error if
   ;    neccessary. A MOV instruction from LS[FE] whth a 3 in the CB field
   ;    will set up the ALU CONDITION CODES. A DECODE instruction with bits 9
   ;    and 4 set to enable IRD STATE L will be issued to set the OPC bits and
   ;    will thereby provide the input to control the PSL CC bits. The PSL CC
   ;    bits can then be tested by issuing a MOV instruction with a 3 in the
   ;    CC field and the MOV instruction from LS[7F] to some WR. In order to
   ;    test OPC 0 and OPC 1 completely all the cases where they're involved
   ;    in setting the PSL CC bits must be checked. For most cases this can be
   ;    done by setting the ALU bits to all zero's and all ones.
   ;
   ;    Each time COMPAT MODE must be set up correctly since MOV instructions
   ;    change its status.The DECODE instructions will load OPCODE REG which
   ;    will allow the OPCODE REG to load the IB BUS when looping on error.
```

```
;
```

```
    ; Assumptions:
    ;
    ;    It is assumed that the logic to load the IB BUS and OPCODE REG has
    ;    been tested and found to be working. It is also assumed that main
    ;    memory has 0 to 256 in it so that all of the OPC2 ROM can be tested.
    ;
    ; Test Steps:
    ;
    ;    1) Set up LS location MM locations and wr's that will be used
    ;    2) Issue a MEM.REQ to fill IB with an incrementing pattern
    ;    3) For COMPAT MODE low (ROM add. 0 - FF) load the Opcode register
    ;       with the data in IB Prefetch
    ;    4) Issue a DECODE with IFUNC odd and the OPCODE DECODE ROM enabled
    ;       (This set IRD STATE)
    ;    5) Get DT CLASS form Size Register, get expected and check
    ;    6) Set the ALU cc's to ones.
    ;    7) Map the cc's to the PSL cc's
    ;    8) Generate expected data according to the CC Class expected
    ;       and check
    ;    9) Repeat steps 6-8 for ALU cc's set to zeroes.
    ;   10) Repeat Steps 2-9 for all ROM addresses from 0 to FF
    ;   11) Set Compat Mode and repeat Steps 2-9 for all ROM addresses
    ;       from 100 to 1FF
    ;
    ; Errors:
    ;
    ;  Other data : ROM address (Note: if OTHER is greater than or equal to
    ;                            100 COMPAT MODE is set and the sencond section
    ;                            of the test is being run)
    ;
    ;    Error 1 - DT Class was not as expected.
    ;    Error 2 - CC Class was not as expected. ALU cc's set.
    ;    Error 3 - CC Class was not as expected. ALU cc's clear.
    ;
    ; Debug:
    ;
    ;    Error 1: The error can be an error in addressing the ROM which would
    ;            result in dual addressing, or the error could be in the con-
    ;            tents of the ROM itself, or the remaining logic is in error.
    ;            The DT CLASS signals are fed into the DATA TYPE CTL PAL and
    ;            come out on the D BUS if a MOV with location 7E in LS is
    ;            executed. They're also loaded into SIZE REG. The DATA TYPE CTL
    ;            PAL and those signals mentioned should be checked for error.
    ;    Error 2-3: This error can result from an error in addressing which would
    ;            cause dual addressing. The error could also be a result of the
    ;            contents of the ROM being incorrect. The error could also
    ;            result from an error in the logic from the output of the OPC2
    ;            ROM to the loading of that information onto the D BUS. Useful
    ;            signals to check are OPC TYPE 1, OPC TYPE 0, CC CLASS 1, CC
    ;            CLASS 0, and the outputs from the PSL CC PAL. The BRANCH COND
    ;            CC CLASS PAL could also be a source of error. The correctness
    ;            of the CC CLASS bits are determined by the PSL bits. The PSL
    ;            bits are a result of the opcode type and the ALU bits.
    ;
    ;--
T.7:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
```

```
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.T7.0:
        JMP [WAIT.T7.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK.
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
;
; T8  = 94 (Pointer to pattern location)
; T9  = Pointer to answer (initially 500)
; T10 = LS storage for pattern
; T11 = Answer Counter
; T12 = Current answer
; T13 = FFFFFFFC Mask for Bits 1 & 0
; T14 = FFFFFFF0 Mask for Bits 3 - 0
; WR2 = WR storgae for pattern
;
        JSR [ASSERT.OTHER]
        CLR LS[ADDRESS.DATA]
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 94
        MOV LS[#400] TO WR[0]
        ADD LS[#100] TO WR[0]
        MOV WR[0] TO LS[T9]             ;T9 = 298
        CLR WR[2]                       ;INITIAL PATTERN = 0
        MOV WR[2] TO LS[T10]
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE INITIAL PATTERN IN MMLOC 94
        CLR LS[PC]
        MOV LS[#FFFFFFFC] TO WR[0]
        MOV WR[0] TO LS[T13]            ;T13 = FFFFFFFC
        MOV LS[#000F] TO WR[0]
        COM WR[0]
        MOV WR[0] TO LS[T14]            ;T14 = FFFFFFF0
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE (NOTE: DATA COMES
                                        ; FROM OPCODE REGISTER)
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T11]         ;T11 = INITIAL COUNT
        INC LS[T9]                      ;INCREMENT MEMORY POINTER
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T12]         ;T12 = FIRST ANSWER
LOOP.T7.1:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB
        DECODE.OPC1 ADRS[T7.JUST.IN.CASE.1],    ;FILL OPCODE REG
         SKIP.IF(IB VALID)
T7.JUST.IN.CASE.1:
        NOP
;
; The following Decode Instruction sets the DT & CC lines from the OPC2 ROM
;  since IRD STATE is asserted (IFUNC odd & Opcode Decode ROM enabled).
```

```
;
        OPC2/DECODE,DEC.ADRS/0,IFUNC/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        MOV LS[SIZE] TO WR[0]           ;GET DT CLASS 0 & DT CLASS 1
        BIC LS[T13] TO WR[0]            ;LOOK ONLY AT BITS 1 & 0
        JSR [T7.GET.EXPECTED.DT]        ;GET EXPECTED
        MOV LS[#1] TO WR[3]
        MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR NUMBER = 1
        JSR [CHECK.RESULT]              ;CHECK DT CLASS 0 & 1
        JMP [LOOP.T7.1]                 ;LOOP ON ERROR IF ENABLED
T7.1.1111:
        INC LS[ERROR.NUMBER]            ;ERROR NUMBER = 2
        MOV LS[#000F] TO WR[3]
        MOV WR[3] TO LS[ALU.CC]         ;SET ALL ALU CC'S
        DT(SIZE)&MAP.ALU.CC.TO.PSL      ;MAP ALU CC TO PSL ACCORDING TO CC CLASS
        MOV LS[PSL.CC] TO WR[0]         ;GET MAPPED CC'S
        BIC LS[T14] TO WR[0]            ;LOOK ONLY AT BITS 3 - 0
        JSR [T7.GET.EXPECTED.CC1111]    ;GET EXPECTED MAPPED CC'S
        JSR [CHECK.RESULT]              ;CHECK CC CLASS 0 & 1 WITH DATA OF 1111
        JMP [LOOP.T7.1]                 ;LOOP ON ERROR IF ENABLED
T7.1.0000:
        INC LS[ERROR.NUMBER]            ;ERROR NUMBER = 3
        CLR WR[0]
        MOV WR[0] TO LS[ALU.CC]         ;CLEAR ALL ALU CC'S
        DT(SIZE)&MAP.ALU.CC.TO.PSL      ;MAP ALU CC TO PSL ACCORDING TO CC CLASS
        MOV LS[PSL.CC] TO WR[0]         ;GET MAPPED CC'S
        BIC LS[T14] TO WR[0]            ;LOOK ONLY AT BITS 3 - 0
        JSR [T7.GET.EXPECTED.CC0000]    ;GET EXPECTED MAPPED CC'S
        JSR [CHECK.RESULT]              ;CHECK CC CLASS 0 & 1 WITH DATA OF 0000
        JMP [LOOP.T7.1]                 ;LOOP ON ERROR IF ENABLED
;
        DEC LS[T11]                     ;DECREMENT ANSWER COUNTER
        JSR [T7.CHECK.COUNTER]          ;CHECK COUNTER & GET NEW ANSWER IF 0
        INC WR[2]
        MOV WR[2] TO LS[T10]            ;INC PATTERN
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE NEW PATTERN TO MMLOC 94
        INC LS[ADDRESS.DATA]
        MOV LS[#100] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC            ;PATTERN = 100 ?
        JMP.IF[NEQ] TO [LOOP.T7.1]      ;LOOP BACK IF NOT
T7.2:
        CLR WR[2]
        CLR LS[T10]                     ;RESET PATTERN
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE NEW PATTERN TO MMLOC 94
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE
LOOP.T7.2:
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB
;
; The following Decode Instruction sets the DT & CC lines from the OPC2 ROM
;   since IRD STATE is asserted (IFUNC odd & Opcode Decode ROM enabled).
;
        OPC2/DECODE,DEC.ADRS/0,IFUNC/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
```

```
	NOP
```

```
        MOV LS[SIZE] TO WR[0]           ;GET DT CLASS 0 & DT CLASS 1
        BIC LS[T13] TO WR[0]            ;LOOK ONLY AT BITS 1 & 0
        JSR [T7.GET.EXPECTED.DT]        ;GET EXPECTED
        MOV LS[#1] TO WR[3]
        MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR NUMBER = 1
        JSR [CHECK.RESULT]              ;CHECK DT CLASS 0 & 1
        JMP [LOOP.T7.2]                 ;LOOP ON ERROR IF ENABLED
T7.2.1111:
        INC LS[ERROR.NUMBER]            ;ERROR NUMBER = 2
        MOV LS[#000F] TO WR[3]
        MOV WR[3] TO LS[ALU.CC]         ;SET ALL ALU CC'S
        DT(SIZE)&MAP.ALU.CC.TO.PSL      ;MAP ALU CC TO PSL ACCORDING TO CC CLASS
        MOV LS[PSL.CC] TO WR[0]         ;GET MAPPED CC'S
        BIC LS[T14] TO WR[0]            ;LOOK ONLY AT BITS 3 - 0
        JSR [T7.GET.EXPECTED.CC1111]    ;GET EXPECTED MAPPED CC'S
        JSR [CHECK.RESULT]              ;CHECK CC CLASS 0 & 1 WITH DATA OF 1111
        JMP [LOOP.T7.2]                 ;LOOP ON ERROR IF ENABLED
T7.2.0000:
        INC LS[ERROR.NUMBER]            ;ERROR NUMBER = 3
        CLR WR[0]
        MOV WR[0] TO LS[ALU.CC]         ;CLEAR ALL ALU CC'S
        DT(SIZE)&MAP.ALU.CC.TO.PSL      ;MAP ALU CC TO PSL ACCORDING TO CC CLASS
        MOV LS[PSL.CC] TO WR[0]         ;GET MAPPED CC'S
        BIC LS[T14] TO WR[0]            ;LOOK ONLY AT BITS 3 - 0
        JSR [T7.GET.EXPECTED.CC0000]    ;GET EXPECTED MAPPED CC'S
        JSR [CHECK.RESULT]              ;CHECK CC CLASS 0 & 1 WITH DATA OF 0000
        JMP [LOOP.T7.2]                 ;LOOP ON ERROR IF ENABLED
;
        DEC LS[T11]                     ;DECREMENT ANSWER COUNTER
        JSR [T7.CHECK.COUNTER]          ;CHECK COUNTER & GET NEW ANSWER IF 0
        INC WR[2]
        MOV WR[2] TO LS[T10]            ;INC PATTERN
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE NEW PATTERN TO MMLOC 94
        INC LS[ADDRESS.DATA]
        MOV LS[#100] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC            ;PATTERN = 100 ?
        JMP.IF[NEQ] TO [LOOP.T7.2]      ;LOOP BACK IF NOT
        JMP [T7.CLEANUP]                ;JUMP OVER SUBS
;
;This subroutine takes bits 2 & 3 of the answer & puts them into bits 0 & 1
; (respectively) of WR1
;
T7.GET.EXPECTED.DT:
        CLR WR[1]                       ;CLEAR EXPECTED
        MOV LS[T12] TO WR[3]            ;STORE ANSWER IN WR3
        BIT LS[BIT2] WITH WR[3],
         DT(LONG)&SET.ALU.CC            ;BIT 2 SET ?
        SKIP.IF[EQL]                    ;SKIP IF NOT
        BIS LS[BIT0] TO WR[1]           ;SET BIT 0
        BIT LS[BIT3] WITH WR[3],
         DT(LONG)&SET.ALU.CC            ;BIT 3 SET ?
        SKIP.IF[EQL]                    ;SKIP IF NOT
        BIS LS[BIT1] TO WR[1]           ;SET BIT 1
        RETURN
;
```

```
;This subroutine checks if the answer counter is zero and if it is gets a new
```

```
; counter from memory (put into LS[T11]) and a new answer (LS[T12]).
;
T7.CHECK.COUNTER:
        MOV LS[T11] TO WR[3]            ;PUT COUNTER IN WR3
        TST WR[3],
         DT(LONG)&SET.ALU.CC            ;COUNTER = 0 ?
        SKIP.IF[EQL]                    ;SKIP IF SO
        RETURN                          ;RETURN IF NOT
        INC LS[T9]                      ;INC POINTER
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T11]         ;GET NEW COUNTER
        INC LS[T9]                      ;INC POINTER
        MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
        MOV MEM.DATA TO LS[T12]         ;GET NEW ANSWER
        RETURN
;
;This subroutine generates in WR1 one of four possible mapped patterns of the
;ALU cc's when all cc's are set
;
T7.GET.EXPECTED.CC1111:
        MOV LS[T12] TO WR[3]            ;GET ANSWER
        MOV LS[#FFFFFFFC] TO WR[1]
        BIC WR[1] TO WR[3]              ;STRIP OFF ALL BUT BITS 1&0
        MOV LS[ZERO] TO Q
        CMP WR[3] WITH Q,
         DT(LONG)&SET.ALU.CC            ;WR3 = 0 ?
        JMP.IF[NEQ] TO [T7.1111.TRY.1]  ;JUMP IF NOT
        MOV LS[#000F] TO WR[1]          ;EXPECTED = ALL CC'S SET
        RETURN
T7.1111.TRY.1:
        MOV LS[#1] TO Q
        CMP WR[3] WITH Q,
         DT(LONG)&SET.ALU.CC            ;WR3 = 1 ?
        JMP.IF[NEQ] TO [T7.1111.TRY.2]  ;JUMP IF NOT
        MOV LS[#000F] TO WR[1]
        BIC LS[BIT0] TO WR[1]           ;EXPECTED = ALL SET BUT C
        RETURN
T7.1111.TRY.2:
        MOV LS[#2] TO Q
        CMP WR[3] WITH Q,
         DT(LONG)&SET.ALU.CC            ;WR3 = 2 ?
        JMP.IF[NEQ] TO [T7.1111.MUST.BE.3]      ;JUMP IF NOT
        CLR WR[1]
        BIS LS[BIT2] TO WR[1]           ;EXPECTED = ALL CLEAR BUT Z
        RETURN
T7.1111.MUST.BE.3:
        MOV LS[#000F] TO WR[1]
        BIC LS[BIT1] TO WR[1]           ;EXPECTED = ALL SET BUT V
        BIC LS[BIT0] TO WR[0]           ;DON'T CARE ABOUT C BIT
        BIC LS[BIT0] TO WR[1]           ;DON'T CARE ABOUT C BIT
        RETURN
;
;This subroutine generates in WR1 one of four possible mapped patterns of the
;ALU cc's when all cc's are clear
;
T7.GET.EXPECTED.CC0000:
        MOV LS[T12] TO WR[3]            ;GET ANSWER
MOV LS[#FFFFFFFC] TO WR[1]
```

```
        BIC WR[1] TO WR[3]              ;STRIP OFF ALL BUT BITS 1&0
        MOV LS[ZERO] TO Q
        CMP WR[3] WITH Q,
         DT(LONG)&SET.ALU.CC            ;WR3 = 0 ?
        JMP.IF[NEQ] TO [T7.0000.TRY.1]  ;JUMP IF NOT
        CLR WR[1]                       ;EXPECTED = ALL CC'S CLEAR
        RETURN
T7.0000.TRY.1:
        MOV LS[#1] TO Q
        CMP WR[3] WITH Q,
         DT(LONG)&SET.ALU.CC            ;WR3 = 1 ?
        JMP.IF[NEQ] TO [T7.0000.TRY.2]  ;JUMP IF NOT
        MOV LS[BIT0] TO WR[1]           ;EXPECTED = ALL CLEAR BUT C
        RETURN
T7.0000.TRY.2:
        MOV LS[#2] TO Q
        CMP WR[3] WITH Q,
         DT(LONG)&SET.ALU.CC            ;WR3 = 2 ?
        JMP.IF[NEQ] TO [T7.0000.MUST.BE.3]      ;JUMP IF NOT
        MOV LS[BIT0] TO WR[1]           ;EXPECTED = ALL CLEAR BUT C
        RETURN
T7.0000.MUST.BE.3:
        CLR WR[1]                       ;EXPECTED = ALL CC'S CLEAR
        BIC LS[BIT0] TO WR[0]           ;DON'T CARE ABOUT C BIT
        RETURN
T7.CLEANUP:
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
END.T7:
    .Page "TEST 8 - Clock Stall Test (M8390 CPU MODULE)"
    ;++
    ;
    ;
    ; Test Description:
    ;
    ;     This test checks the enable of CLOCK STALL. A clock stall will occur
    ;     if a DECODE instruction is issued with PC=3, which causes a memory
    ;     request followed by another DECODE instruction. This stall occurs
    ;     because the first DECODE instruction caused a memory request which
    ;     wasn't completed by the time the second DECODE instruction was issued.
    ;     Error  can be detected by having different data present in IB PREFETCH
    ;     and in the next memory location.
    ;
    ; Assumptions:
    ;
    ;     It's assumed that the IB PREFETCH REG and the OS REG have been checked.
    ;     It's also assumed that the DECODE instruction works and a DECODE with
    ;     PC=3 will initiate a memory request. It's assumed that the PREFETCH
    ;     REG has zeros in it and the main memory has ones.
    ;
    ; Test Steps:
    ;
    ;     1) Issue a DECODE instruction with PC=3 and IB REQUEST set which will
    ;        initiate a memory request. Allow time for memory to finish up.
    ;     2) Issue a second DECODE instruction to load the OS REG.
    ;     3) Load the OS REG into WR[0] and ones into WR[1] and call Check
    ;        Result.
;       4) Issue a DECODE instruction with PC=3 and IB REQUEST set which will
```

```
   ;       initiate a memory request. Allow time for memory to finish up.
   ;    5) Issue a second DECODE instruction to load the OS REG.
   ;    6) Load the OS REG into WR[0] and ones into WR[1] and call Check
   ;       Result.
   ;
   ; Error:
   ;
   ;    Error 1 - IB.VALID not deasserted during memory request.
   ;    Error 2 - CLOCK STALL failed to be enabled.
   ;
   ; Debug:
   ;
   ;    Error 1: If IB.Valid was not deasserted during the memory request
   ;            check the inputs to the IB IR CONTROL PAL. Specifically
   ;            0 & 1 of the Y bus, DEC INST, and CPU P2.
   ;    Error 2: If CLOCK STALL failed there are two areas of the logic that
   ;            could be the source of error. The first is the gates producing
   ;            CLOCK STALL. Of these gates it's likely that STALL ON IB would
   ;            be the gate in error if there were an error. The second pos-
   ;            sible source of error comes from the IB IR CONTROL PAL.
   ;
   ;--
T.8:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.T8.0:
        JMP [WAIT.T8.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        MOV LS[#000F] TO WR[0]
        MOV LS[#00F0] TO WR[2]
        ADD WR[2] TO WR[0]
        COM WR[0]
        MOV WR[0] TO LS[ERROR.MASK]     ;MASK = FFFFFF00
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        MOV LS[#80] TO WR[0]
        ADD LS[#8] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 88 Pointer to word of zeroes
        ADD LS[#2] TO WR[0]
        MOV WR[0] TO LS[T9]             ;T9 = 8A
LOOP.T8.1:
        MOV LS[T9] TO WR[0]
        MOV WR[0] TO LS[PC]             ;PC = 8A
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH ZEROES
        NOP                             ;ALLOW TIME FOR MEMORY TO FINISH
        NOP                             ;               |
        NOP                             ;               |
        NOP                             ;               |
        NOP                             ;              -V-
        MOV IB.DATA TO OS               ;MOV IB DATA TO OS & INC PC
        NOP
        MOV IB.DATA TO OS                   ;MOV IB DATA TO OS & INC PC
        NOP
T8.1:
        MOV IB.DATA TO OS               ;GET NEXT BYTE
                                        ; INITIATED ABOVE IS FINISHED)
        JMP [T8.1]                      ;IF NOT IB.VALID JUMP BACK
        MOV LS[OS] TO WR[0]             ;GET WHAT WAS WRITTEN
        MOV LS[#000F] TO WR[1]
        MOV LS[#00F0] TO WR[2]
        ADD WR[2] TO WR[1]              ;EXPECTED = 000000FF
        JSR [CHECK.RESULT]              ;CHECK THAT CLOCK STALL WORKED
        JMP [LOOP.T8.1]
LOOP.T8.2:
        MOV LS[T9] TO WR[0]
        MOV WR[0] TO LS[PC]             ;PC = 8A
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH ZEROES
        MOV IB.DATA TO OS               ;MOV IB DATA TO OS & INC PC (SHOUL STALL UNTIL MEMORY FINISHEC)
        NOP
        MOV IB.DATA TO OS               ;MOV IB DATA TO OS & INC PC (SHOUL STALL UNTIL MEMORY FINISHEC)
        NOP
T8.2:
        MOV IB.DATA TO OS               ;GET NEXT BYTE
                                        ; INITIATED ABOVE IS FINISHED)
        JMP [T8.2]                      ;IF NOT IB.VALID JUMP BACK
        MOV LS[OS] TO WR[0]             ;GET WHAT WAS WRITTEN
        MOV LS[#000F] TO WR[1]
        MOV LS[#00F0] TO WR[2]
        ADD WR[2] TO WR[1]              ;EXPECTED = 000000FF
        JSR [CHECK.RESULT]              ;CHECK THAT CLOCK STALL WORKED
        JMP [LOOP.T8.2]
END.T8:
    .Page "TEST 9 - Register Mode And GPR Dest Test (M8390 CPU MODULE)"
    ;++
    ;
    ;
    ; Test Description:
    ;
    ;     This test checks to see that the logic that manipulates the REGISTER
    ;     MODE and the GPR DEST signals is functioning correctly. REGISTER MODE
    ;     and GPR DEST will both be asserted under one of two conditions. They
    ;     will be asserted if IB bits 4 - 7 have 0101 in them and the instruc-
    ;     tion is a DECODE with bits 5 and 6 set. The other possibility is if
    ;     the instruction is a DECODE with bits 5 and 6 set and COMPAT MODE is
    ;     asserted and IB BUS bits 3, 4, and 5 are clear. In order to test these
    ;     cases fully patterns similar to the patterns that will set REGISTER
    ;     MODE will be tried to insure that only the correct pattern asserts
    ;     REGISTER MODE. The patterns that will be tested in VAX MODE are 0101,
    ;     1101, 0001, 0111, and 0100. The patterns to be tested in COMPAT MODE
    ;     are 0101, 000 in bits 3 - 5, 001, 010, and 100.
    ;
    ; Assumptions:
    ;
    ;     It is assumed that the IB BUS has been tested and works and the enable
    ;     to and the SPEC DECODE ROM have been tested and also work correctly.
    ;
    ; Test Steps:
    ;
    ;     1) Issue a DECODE instruction with bits 5 and 7 set and a 50 on the IB
;       BUS.
;     2) Issue a NOP with a skip on RDEST in the skip control field. If
;        there's no skip then print errror1.
;     3) Else Issue a DECODE instruction with bits 5 and 7 set and one of
;        the four "wrong" patterns to be tested on the IB BUS.
;     4) Issue a NOP with a skip on RDEST in the skip control field. If
;        there is a skip then print error2.
;     5) Else if not all "wrong" patterns tested then jump to step 3.
;     6) Else issue a MOV instruction to set COMPAT MODE.
;     7) Issue a DECODE instruction with bits 5 and 7 set and one of 4
;        "wrong" patterns on the IB BUS.
;     8) Issue a NOP instruction with a skip on RDEST in the skip control
;        field if skip the print error3, else no error.
;     9) If no error and all "wrong" patterns not tested then jump to step
;        6 else issue a MOV instruction to assert COMPAT MODE.
;      10) Issue a DECODE instruction with bits 5 and 6 set and 0 on the IB
;       BUS.
;      11) Issue a NOP with a skip on RDEST in the skip control field, if no
;        skip then issue error4.
;
; Error:
;
;     Error 1 - R MODE asserted in VAX mode failed.
;     Error 2 - R MODE unasserted in VAX mode failed.
;     Error 3 - R MODE unasserted in COMPAT MODE failed.
;     Error 4 - R MODE asserted in COMPAT MODE failed.
;
; Debug:
;
;     Error 1: If there's an error then GPR DEST wasn't asserted when it
;              should have been. There are several possible sources of error.
;              This first is with SEQ CTL PAL and the SEQ CTL MUX. If GPR
;              DEST is asserted an this logic is faulty then there will be
;              no skip produced. The next possible source of error is the
;              CM RDEST PAL which outputs GPR DEST. Of the inputs that define
;              GPR DEST the only one that is likely to be in error is REGIS-
;              TER MODE. REGISTER MODE comes from the BRANCH COND CC CLASS
;              PAL and is defined by RMODE A and RMODE B. RMODE A is defined
;              in the OS <7:1>(ODD) PAL and RMODE B is defined by the OS
;              <6:0>(EVEN) PAL. RMODE A and RMODE B are defined such that if
;              if there are certain patterns on the IB BUS then RNODE A and
;              RMODE B are asserted.
;     Error 2: IF this error occurs then a pattern that shouldn't assert GPR
;              DEST does. This kind of error usually indicates a short be-
;              tween pins, although any of the logis mentioned in error 1
;              could also be in error.
;     Error 3: In this case GPR DEST should be unasserted as a result of the
;              the wrong pattern being loaded onto the IB BUS when the macine
;              is in COMPAT MODE. As in in the case of error 2 if there's an
;              error it's probably the result a short between pins, but can
;              also be the result of an error in any of the logic mentioned
;              in error 1.
;     Error 4: If this error occurs then GPR DEST should be asserted, the
;              machine should be in COMPAT MODE and there should be 0's in
;              bits 3 - 5 of the IB BUS. The possible sources of error in
;              this case are the same as those mentioned in error 1 and these
;              should be checked for error.
;--
```

T.9:

```
        MOV LS[BEGIN.TEST] TO WR[0]       ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS]   ;SET BIT15 IN CNTROL AND STATUS WORD
                                          ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]                ;ISSUE CPU ATTN TO CONSOLE
WAIT.T9.0:
        JMP [WAIT.T9.0]                   ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]           ;CLEAR COMPAT MODE
        CLR LS[ERROR.MASK]                ;CLEAR ERROR MASK.
        CLR LS[PREVIOUS.ERROR]            ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]               ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]     ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                         ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]       ;SET UP MODULE CODE FOR CPU MODULE
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]               ;T8 = 94 Pointer to Pattern Location
        CLR LS[T10]                       ;T10 = PATTERN
        JSR [ASSERT.NA]                   ;NO EXPECTED OR RECEIVED DATA
LOOP.T9.1:
        CLR LS[PC]
        MOV LS[BIT6] TO WR[0]
        BIS LS[BIT4] TO WR[0]
        MOV WR[0] TO LS[T10]              ;PATTERN = 00000050
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                 ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;PUT PATTERN ON IB BUS
        OPC2/DECODE,IB.REQ/1,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                    ;SKIP IF RDEST WAS ENABLED
        SKIP
        JMP [T9.2]                        ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]             ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]                ;REPORT ERROR
        JMP [LOOP.T9.1]                   ;LOOP ON ERROR IF ENABLED
T9.2:
        INC LS[ERROR.NUMBER]              ;INC ERROR NUMBER TO 2
LOOP.T9.2.0001:
        MOV LS[BIT4] TO WR[0]
        MOV WR[0] TO LS[T10]              ;PATTERN = 00000010
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                 ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;PUT PATTERN ON IB BUS
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                    ;SKIP IF RDEST WAS ENABLED
        JMP [LOOP.T9.2.0100]              ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]             ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]                ;REPORT ERROR
        JMP [LOOP.T9.2.0001]              ;LOOP ON ERROR IF ENABLED
LOOP.T9.2.0100:
        MOV LS[BIT6] TO WR[0]
        MOV WR[0] TO LS[T10]              ;PATTERN = 00000040
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                    ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;PUT PATTERN ON IB BUS
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                  ;SKIP IF RDEST WAS ENABLED
        JMP [LOOP.T9.2.0111]            ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]           ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.T9.2.0001]            ;LOOP ON ERROR IF ENABLED
LOOP.T9.2.0111:
        MOV LS[BIT6] TO WR[0]
        BIS LS[BIT5] TO WR[0]
        BIS LS[BIT4] TO WR[0]
        MOV WR[0] TO LS[T10]            ;PATTERN = 00000070
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;PUT PATTERN ON IB BUS
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                  ;SKIP IF RDEST WAS ENABLED
        JMP [LOOP.T9.2.1101]            ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]           ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.T9.2.0001]            ;LOOP ON ERROR IF ENABLED
LOOP.T9.2.1101:
        MOV LS[BIT6] TO WR[0]
        BIS LS[BIT7] TO WR[0]
        BIS LS[BIT4] TO WR[0]
        MOV WR[0] TO LS[T10]            ;PATTERN = 000000D0
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;PUT PATTERN ON IB BUS
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                  ;SKIP IF RDEST WAS ENABLED
        JMP [T9.3]                      ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]           ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.T9.2.0001]            ;LOOP ON ERROR IF ENABLED
T9.3:
        INC LS[ERROR.NUMBER]            ;INC ERROR NUMBER TO 3
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE
LOOP.T9.3.001:
        MOV LS[BIT3] TO WR[0]
        MOV WR[0] TO LS[T10]            ;PATTERN = 00000008
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;PUT PATTERN ON IB BUS
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                  ;SKIP IF RDEST WAS ENABLED
        JMP [LOOP.T9.3.010]             ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]           ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]                 ;REPORT ERROR
        JMP [LOOP.T9.3.001]                ;LOOP ON ERROR IF ENABLED
LOOP.T9.3.010:
        MOV LS[BIT4] TO WR[0]
        MOV WR[0] TO LS[T10]               ;PATTERN = 00000010
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                  ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]        ;PUT PATTERN ON IB BUS
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                     ;SKIP IF RDEST WAS ENABLED
        JMP [LOOP.T9.3.100]                ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]              ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]                 ;REPORT ERROR
        JMP [LOOP.T9.3.010]                ;LOOP ON ERROR IF ENABLED
LOOP.T9.3.100:
        MOV LS[BIT3] TO WR[0]
        MOV WR[0] TO LS[T10]               ;PATTERN = 00000020
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                  ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]        ;PUT PATTERN ON IB BUS
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                     ;SKIP IF RDEST WAS ENABLED
        JMP [T9.4]                         ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]              ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]                 ;REPORT ERROR
        JMP [LOOP.T9.3.100]                ;LOOP ON ERROR IF ENABLED
T9.4:
        INC LS[ERROR.NUMBER]               ;ERROR NUMBER = 4
LOOP.T9.4:
        CLR WR[0]
        MOV WR[0] TO LS[T10]               ;PATTERN = 00000000
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                  ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]        ;PUT PATTERN ON IB BUS
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[R.DST]                     ;SKIP IF RDEST WAS ENABLED
        SKIP
        JMP [END.T9]                       ;GO ON IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]              ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]                 ;REPORT ERROR
        JMP [LOOP.T9.4]                    ;LOOP ON ERROR IF ENABLED
END.T9:
    .Page "TEST A - OS Control Test (M8390 CPU MODULE)"
    ;++
    ;
    ;
    ; Test Description:
    ;
    ;     This test checks to see that OS CTL 1 and OS CTL 0 are not stuck at
    ;     zero or one. A DECODE instruction with the LOAD OS bit set and the
    ;     LOAD RDEST bit clear causes OS CTL 1 and OS CTL 0 to be low. This
```

```
;	should cause the OS REG to be loaded from the IB BUS. If the OS REG
```

```
;       contains what was in IB PREFETCH then OS CTL 1 and OS CTL 0 work as
;       expected at zero. A MOV WR[0] to LS[OS] will cause OS CTL 0 and OS CTL
;       1 to be one. It should also cause the OS REG to be loaded with the
;       contents of WR[0]. A DECODE instruction with R DEST set and LOAD OS
;       set will cause OS CTL 0 to be 0 and OS CTL 1 to be 1 and the OS to be
;       loaded from the IB BUS. In order to set OS CTL 1 equal to 0 and OS CTL
;       0 equal to 1 a MOV instruction with bit 31 set must be issued to bring
;       COMPAT MODE high, followed by a DECODE instruction to bring CM IRD
;       high followed by a MOV instruction which will load the OS REG with the
;       contents of WR[0] and cause OS CTL 1 to be 0 and OS CTL 0 to be 1. If
;       all instructions work as expected, then OS CTL 0 and OS CTL 1 have
;       been tested for stuck at zero and stuck at one.
;
; Assumptions:
;
;       It's assumed that the OS REG and the IB BUS and the IB PREFETCH REG
;       have all been tested and found to be working. Also main memory
;       should contain ones and WR[2] should have zero in it.
;
; Test Steps:
;
;       1) A DECODE instruction with RDEST clear and LOAD OS set will cause OS
;          CTL 0 and OS CTL 1 to 0 and the OS REG to be loaded from the IB BUS
;       2) The OS REG is loaded into WR[0] and WR[1] is loaded with ones and
;          Check Result is called.
;       3) If an error and loop on error then go to step 1.
;       4) Else issue a MOV instruction from WR[2] to LS[OS] which causes OS
;          CTL 1 and OS CTL 0 to be 1 and OS REG to be loaded from WR[2].
;       5) The OS REG is loaded into WR[0] and zeros into WR[1] and Check
;          Result is called.
;       6) If an error and loop on error then go to step 4.
;       7) Else issue a DECODE instruction with RDEST set and LOAD OS set
;          causing OS CTL 1 to be 1 and OS CTL 0 to be 0 and the OS REG to
;          be loaded from the IB BUS.
;       8) Load WR[0] from the OS REG and load WR[1] with ones and call Check
;          Result.
;       9) If an error and loop on error then go to step 7.
;         10) Else issue a MOV instruction with bit 31 set to bring COMPAT MODE
;          high then a DECODE instruction to bring CM IRD high for the next
;          clock cycle and then a MOV WR[2] to LS[OS] which causes OS CTL 1 to
;          be 0 and OS CTL 0 to be 1 and the OS REG to be loaded with the
;          contents of WR[2].
;         11) Load WR[0] from the OS REG and load WR[1] with zeros and call Check
;          Result.
;         12) If error and loop on error then go to step 10.
;
; Error:
;
;       Error 1 - Either OS CTL 1 is 1 or OS CTL 0 is 1.
;       Error 2 - Either OS CTL 1 is 0 or OS CTL 0 is 0.
;       Error 3 - Either OS CTL 1 is 0 or OS CTL 0 is 1.
;       Error 4 - Either OS CTL 1 is 1 or OS CTL 0 is 0.
;
; Debug:
;
;       Error 1: If the CM RDEST PAL has been checked then either OS CTL 1
;                or OS CTL 0 is stuck at one, and by checking errors 2 - 4
;               of this test, isolation should be possible.
```

```
   ;    Error 2: Assuming that the CM RDEST PAL has been checked then either
   ;            OS CTL 1 or OS CTL 0 is stuck at zero.
   ;    Error 3: Again if the CM RDEST PAL has been checked then either OS CTL
   ;            1 is stuck at 0 or OS CTL 0 is stuck at 1.
   ;    Error 4: Check to see that CM IRD is high and if not check the CM RDEST
   ;            PAL otherwise either OS CTL 1 is stuck at 1 or OS CTL 0 is
   ;            stuck at 0.
   ;
   ;--
T.A:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.TA.0:
        JMP [WAIT.TA.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK.
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        CLR LS[PC]
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 90 POINTER TO MM WORD WITH BIT 0
                                        ;       OF EACH BYTE SET (01010101 (H))
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;SET BIT 0 OF IB BUS
LOOP.TA.1:
        CLR LS[PC]
        MOV LS[ONES] TO WR[0]
        MOV WR[0] TO LS[OS]             ;PRELOAD OS WITH ONES
        OPC2/DECODE,IB.REQ/1,BPC/NOP,DEC.ADRS/0,IFUNC/0,OPC.SPEC/0,LD.OS/1,R.DST/0,JCTL/NO.JUMP.TST
        NOP
        MOV LS[OS] TO WR[0]             ;OS SHOULD BE LOADED
        MOV LS[BIT0] TO WR[1]           ;EXPECTED = BIT0
        JSR [CHECK.RESULT]              ;CHECK THAT DECODE LOADED OS FROM IB BUS
        JMP [LOOP.TA.1]                 ;LOOP ON ERROR IF ENABLED
TA.2:
        INC LS[ERROR.NUMBER]            ;INC ERROR NUMBER TO 2
LOOP.TA.2:
        CLR LS[PC]
        MOV LS[ONES] TO WR[0]
        MOV WR[0] TO LS[OS]             ;PRELOAD OS WITH ONES
        MOV LS[BIT1] TO WR[2]           ;SET BIT 1 IN WR2
        MOV WR[2] TO LS[OS]             ;LOAD OS
        MOV LS[OS] TO WR[0]
        MOV LS[BIT1] TO WR[1]           ;EXPECTED = BIT1
        JSR [CHECK.RESULT]              ;CHECK THAT OS WAS NOT LOADED FROM IB BUS
        JMP [LOOP.TA.2]                 ;LOOP ON ERROR IF ENABLED
TA.3:
        INC LS[ERROR.NUMBER]            ;INC ERROR NUMBER TO 3
LOOP.TA.3:
        CLR LS[PC]
        MOV LS[ONES] TO WR[0]
        MOV WR[0] TO LS[OS]              ;PRELOAD OS WITH ONES
        OPC2/DECODE,IB.REQ/1,BPC/NOP,DEC.ADRS/0,IFUNC/0,OPC.SPEC/0,LD.OS/1,R.DST/1,JCTL/NO.JUMP.TST
        NOP
        MOV LS[OS] TO WR[0]             ;GET OS
        MOV LS[BIT0] TO WR[1]           ;EXPECTED = BIT0
        JSR [CHECK.RESULT]              ;CHECK THAT OS WAS LOADED FROM THE IB BUS
        JMP [LOOP.TA.3]                 ;LOOP ON ERROR IF ENABLED
TA.4:
        INC LS[ERROR.NUMBER]            ;INC ERROR NUMBER TO 4
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE
LOOP.TA.4:
        CLR LS[PC]
        MOV LS[ONES] TO WR[0]
        MOV WR[0] TO LS[OS]             ;PRELOAD OS WITH ONES
;This DECODE sets IRD for one cycle
        OPC2/DECODE,DEC.ADRS/0,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        MOV WR[2] TO LS[OS]             ;LOAD OS
        MOV LS[OS] TO WR[0]
        CLR WR[1]                       ;EXPECTED = CLEAR (WR2 NOT LOADED)
        JSR [CHECK.RESULT]              ;CHECK THAT OS WAS NOT LOADED FROM IB BUS
        JMP [LOOP.TA.4]                 ;LOOP ON ERROR IF ENABLED
CLEANUP.TA:
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
END.TA:
    .Page "TEST B - OS Bits Test"
    ;++
    ;
    ; Test Description:
    ;
    ;   The purpose of this test is to check the contents of the OS REGISTER
    ;   under different OS CTL's and different data on the IB BUS. When OS CTL
    ;   is 0 then the OS REGISTER is loaded from the IB BUS. This has already
    ;   been tested by rippling 1's through 0's, .When OS CTL is 1 then
    ;   OS 0 gets IB 6, OS 1 gets IB 7, OS 2 gets OS 0, OS 3 gets 0, OS 4 gets
    ;   OS 2, OS 5 gets OS 3, OS 6 gets OS 4, and OS 7 gets OS 5. In order to
    ;   test the OS REGISTER under these circumstances a shifting pattern
    ;   through the IB BUS with the OS REGISTER remaining constant followed by
    ;   a shifting pattern through the OS REGISTER with the IB BUS remaining
    ;   constant. In the case where OS CTL is 2 the same patterns used when
    ;   OS CTL is 1 will test the OS REGISTER. When OS CTL is 3 then you can
    ;   test the OS REGISTER by shifting ones through the OS REGISTER.
    ;
    ; Assumptions:
    ;
    ;   It is assumed that the OS CONTROL TEST has been run successfully and
    ;   that the IB BUS, OS REGISTER, and IB PREFETCH REGISTER have been
    ;   tested and work. Memory should be loaded with shifting one pattern
    ;   which is loaded into the PREFETCH REGISTER.
    ;
    ; Test Steps:
    ;
    ;
    ;   1) Issue a MOV instruction with bit 31 set to bring COMPAT MODE high
    ;      followed by DECODE instrution to assert CM IRD, this will bring
;       OS CTL to 1.
```

```
    ;     2) Issue a MOV instruction from OS to WR[0] and load expected data into
    ;        WR[1] and call Check Result.
    ;     3) Issue a MOV instruction with bit 31 set to bring COMPAT MODE high
    ;        followed by a DECODE instruction to aasert CM IRD and load the IB
    ;        BUS with a shifting pattern.
    ;     4) Issue a MOV instruction to load OS from WR[2] which should contain
    ;        zero's. Load OS into WR[0] and expected data into WR[1] and call
    ;        Check Result. If all shifting patterns tested then go to step 6,
    ;        else go to step 8.
    ;     5) Issue a DECODE instruction with both RDEST and LOAD OS set which
    ;        causes OS CTL to be set to 2 and the OS REGISTER to be loaded from
    ;        the IB BUS.
    ;     6) MOV OS into WR[0] and load the expected data into WR[1] then call
    ;        Check Result. If all shifting patterns tested then go to step 10
    ;        else get next pattern and go to step 8.
    ;
    ; Error:
    ;
    ;     Error 1 - Failure in the OS REGISTER when OS CTL is 1.
    ;     Error 2 - Failure in the OS REGISTER when OS CTL is 2.
    ;
    ; Debug:
    ;
    ;     Error 1: If this error occurs then one of the patterns used to test
    ;              the OS REGISTER when OS CTL is 1 has failed. By examining the
    ;              expected and received data one can determine the pattern that
    ;              failed and where it failed. Since the OS CTL logic has been
    ;              tested previously the most likely source of error is the OS
    ;              <7:1>(ODD) PAL if the failed bit is odd and the OS<6:0>(EVEN)
    ;              PAL if the failed bit is even.
    ;     Error 2: If this error occurs then one of the patterns used to test the
    ;              OS REGISTER when OS CTL is 2 has failed. By examining the ex-
    ;              pected and received data one can determine the pattern that
    ;              failed and where it failed. Since the OS CTL logic has been
    ;              tested previously the most likely source of error is the OS
    ;              <7:1>(ODD) PAL if the failed bit is odd and the OS<6:0>(EVEN)
    ;              PAL if the failed bit is even.
    ;
    ;--
T.B:
        MOV LS[BEGIN.TEST] TO WR[0]       ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS]   ;SET BIT15 IN CNTROL AND STATUS WORD
                                          ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]                ;ISSUE CPU ATTN TO CONSOLE
WAIT.TB.0:
        JMP [WAIT.TB.0]                   ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MCOM LS[#FF] TO WR[0]
        MOV WR[0] TO LS[ERROR.MASK]       ;ERROR MASK = FFFFFF00
        CLR LS[PREVIOUS.ERROR]            ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]               ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]     ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                         ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]       ;SET UP MODULE CODE FOR CPU MODULE
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]               ;T8 = 94 POINTER TO MEMORY WORD
        MOV LS[BIT8] TO WR[2]           ;INIT PATTERN
        MOV LS[HI.IPL] TO WR[0]
        BIS LS[BIT31] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;SET COMPAT MODE
LOOP.TB.1:
        CLR LS[PC]
        MOV WR[2] TO LS[T10]            ;WRITE PATTERN TO LS
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;PUT PATTERN ON IB BUS
        MOV LS[ONES] TO WR[0]
        MOV WR[0] TO LS[OS]             ;PREFILL OS WITH 1'S
;
;The following DECODE sets OS CTL to 01 (B)
;
        OPC2/DECODE,IB.REQ/1,BPC/NOP,DEC.ADRS/0,IFUNC/1,OPC.SPEC/1,JCTL/NO.JUMP.TST
        NOP
        NOP                             ;ALLOW TIME FOR OS TO GET CHANGED
        MOV LS[OS] TO WR[0]             ;GET OS
        JSR [TB.GEN.EX]                 ;GENERATE EXPECTED DATA
        JSR [CHECK.RESULT]              ;CHECK OS
        JMP [LOOP.TB.1]                 ;LOOP ON ERROR IF ENABLED
        MOV LS[BIT15] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC            ;PATTERN = 00008000 ?
        JMP.IF[EQL] TO [TB.2]           ;GO ON IF SO
        ROL WR[2]                       ;SHIFT PATTERN
        JMP [LOOP.TB.1]                 ;LOOP BACK
TB.2:
        INC LS[ERROR.NUMBER]            ;INC ERROR NUMBER TO 2
        MOV LS[BIT0] TO WR[2]           ;RESET PATTERN
LOOP.TB.2:
        MOV WR[2] TO LS[T10]            ;WRITE PATTERN TO LS
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;PUT PATTERN ON IB BUS
        MOV LS[ONES] TO WR[0]
        MOV WR[0] TO LS[OS]             ;PREFILL OS WITH 1'S
;
;The following DECODE sets OS CTL to 10  (B)
;
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/0,LD.OS/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        MOV LS[OS] TO WR[0]             ;GET OS
        MOV WR[2] TO WR[1]
        BIC LS[BIT3] TO WR[1]           ;EXPECTED = PATTERN EXCEPT BIT3 CLEARED
        JSR [CHECK.RESULT]              ;CHECK OS
        JMP [LOOP.TB.2]                 ;LOOP ON ERROR IF ENABLED
        ROL WR[2]                       ;SHIFT PATTERN
        MOV LS[BIT8] TO Q
        CMP WR[2] WITH Q,
         DT(LONG)&SET.ALU.CC            ;PATTERN = 00000100 ?
        JMP.IF[NEQ] TO [LOOP.TB.2]      ;LOOP BACK IF NOT
        JMP [CLEANUP.TB]
TB.GEN.EX:
        MOV WR[2] TO WR[3]              ;WR3 = PATTERN
        MOV LS[ONES] TO WR[1]
        BIC LS[BIT0] TO WR[1]
BIC LS[BIT1] TO WR[1]
```

```
        BIC LS[BIT3] TO WR[1]           ;CLEAR BITS 3, 1, & 0
        BIT LS[BIT14] WITH WR[3],
         DT(LONG)&SET.ALU.CC            ;BIT 6 SET IN WR3 ?
        SKIP.IF[EQL]                    ;SKIP IF NOT
        BIS LS[BIT0] TO WR[1]           ;SET BIT 0 IN WR1 IF SO
        BIT LS[BIT15] WITH WR[3],
         DT(LONG)&SET.ALU.CC            ;BIT 7 SET IN WR3 ?
        SKIP.IF[EQL]                    ;SKIP IF NOT
        BIS LS[BIT1] TO WR[1]           ;SET BIT 1 IN WR1 IF SO
        RETURN
CLEANUP.TB:
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
END.TB:
    .Page "TEST C - GPR Dest Test (M8390 CPU MODULE)"
    ;++
    ;
    ;
    ; Test Description:
    ;
    ;     This test checks the GPR DEST signal for stuck at zero and stuck at
    ;     one. There are two parts to checking the signal; the first being to
    ;     set GPR DEST to either one or zero, and the second to check to see if
    ;     GPR DEST is one or zero. To bring GPR DEST low (asserted) the instruc-
    ;     tion must be a DECODE with the RDEST bit (CSR 05) set and bits 4 and 6
    ;     of the IB BUS must be set or bits 5 and 7 of the IB BUS must be clear.
    ;     Once the DECODE instruction has been issued to set GPR DEST low or
    ;     high then there must be a way to test to see if GPR DEST is what it's
    ;     supposed to be. Within the DECODE instruction there is a jump control
    ;     field which causes a jump to someplace in WCS if the condition is met.
    ;     One of these conditions is RDEST which jumps if GPR DEST is set. It is
    ;     also possible to skip on RDEST and this will also be done to insure
    ;     that skip on RDEST is working.
    ;
    ; Assumptions:
    ;
    ;     It's assumed the SPEC DECODE ROM is tested and the OPC2 ROM is tested
    ;     and the logic following those ROMs which cause GPR DEST to be enabled
    ;     is tested.
    ;
    ; Test Steps:
    ;
    ;     1) Clear WR[3] and issue a DECODE instruction with RDEST set and 6 in
    ;        the jump control field. Zero's in IB Prefetch and a DECODE .SPEC
    ;        instruction that will cause a jump to the address specified by the 1ST
    ;        location in the SPEC DECODE ROM.
    ;     2) If there's an error then there will be no jump so WR3 should be
    ;        changed. (The JSR is to the block of 256 INC WR[3]'s)
    ;     3) Clear WR[3] and issue a DECODE instruction with RDEST clear and 6 in
    ;        the jump control field. Zero's in IB Prefetch and a DECODE .SPEC
    ;        instruction will cause a jump to the address specified by the 1ST
    ;        location in the SPEC DECODE ROM if RDEST is still there.
    ;     4) If there's an error then there will be a jump so WR3 should be
    ;        unchanged. (The JSR is to the block of 256 INC WR[3]'s)
    ;
    ; Error:
    ;
```

```
;	Error 1 - GPR DEST is stuck at zero or the jump control failed
```

```
   ;    Error 2 - GPR DEST is stuck at one or the jump control for RDEST failed
   ;
   ; Debug:
   ;
   ;    Error 1: The DECODE instruction should have caused a jump to the WCS.
   ;            The logic leading up to the CM RDEST PAL is known to be cor-
   ;            rect so places to look for error are the CM RDEST PAL, the
   ;            SEQ CONTROL MUX and the SEQ CONTROL PAL. If the signal is not
   ;            in error then it's the jump control logic. This can be easily
   ;            determined by checking to see if both error 1 and error 2 come
   ;            up. If this is the case then the jump control logic is in
   ;            error.
   ;    Error 2: The possible sources of error are the CM RDEST PAL which would
   ;            cause GPR DEST to be incorrect and the SEQ CONTROL MUX and SEQ
   ;            CONTROL PAL which would cause jump control logic to be incor-
   ;            rect. If the CM RDEST PAL is alright then the jump control
   ;            logic could be wrong or GPR DEST could be stuck at zero. The
   ;            jump control logic would be in error if both error 1 and error
   ;            2 came up otherwise it would be GPR DEST.
   ;
   ;--
T.C:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.TC.0:
        JMP [WAIT.TC.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK.
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        JSR [ASSERT.NA]
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 94 POINTER TO MEMORY LOCATION
        CLR LS[T10]                     ;CLEAR LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;CLEAR MEM LOC 94
LOOP.TC.1:
        CLR LS[PC]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;CLEAR IB
        OPC2/DECODE,DEC.ADRS/7,BPC/NOP,IFUNC/0,IB.REQ/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[R.DST] TO [TC.2]         ;JUMP IF R DEST SET
        CLR WR[0]                       ;REPORT ERROR IF NOT
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]
        JMP [LOOP.TC.1]
TC.2:
        INC LS[ERROR.NUMBER]            ;INC ERROR NUMBER TO 2
        MOV LS[#10] TO WR[0]
```

```
ADD LS[#20] TO WR[0]
```

```
        MOV WR[0] TO LS[T10]             ;T10 = 30
LOOP.TC.2:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                ;MEM LOC 94 = 30
        CLR LS[PC]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]       ;IB = 30
        OPC2/DECODE,DEC.ADRS/7,BPC/NOP,IFUNC/0,IB.REQ/1,R.DST/1,OPC.SPEC/0,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[R.DST] TO [TC.2.ERR]      ;JUMP IF RDEST STILL THERE
        JMP [END.TC]                     ;GO TO END IF NOT
TC.2.ERR:
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]               ;REPORT ERROR
        JMP [LOOP.TC.2]
END.TC:
   .Page "TEST D - Register Backup Flag Test (M8390 CPU MODULE)"
   ;++
   ;
   ;
   ; Test Description:
   ;
   ;    This test checks the signal RBKUP FLAG for stuck at zero and stuck
   ;    at one. A MISC instruction will set the RBKUP FLAG bit. The state
   ;    of the RBKUP FLAG bit can be tested by skipping on REG BACKUP MASK
   ;    FLAG. If it skips then the bit is set which is what is expected
   ;    and therefore it's known that RBKUP FLAG is not stuck low. The
   ;    RBKUP FLAG bit can be cleared by issuing a DECODE instruction with
   ;    bits 9 and 4 of the CSR set. In order to test the RBKUP FLAG to
   ;    see if it's clear issue a NOP with the skip condition set to skip
   ;    on REG BACKUP MASK FLAG. If it skips then RBKUP FLAG is stuck high.
   ;
   ; Assumptions:
   ;
   ;    It is assumed that IB BUS has been tested and works.
   ;
   ; Test Steps:
   ;
   ;    1) Initialize RBKUP FLAG to zero by issuing a DECODE instruction
   ;       with bits 9 and 4 set.
   ;    2) In order to set RBKUP FLAG high, MISC instruction with MF1 set to
   ;       seven is issued.
   ;    3) Using the skip condition REG BACKUP MASK FLAG and a NOP it can be
   ;       dicovered whether or not RBKUP FLAG is high or not.
   ;    4) If a skip occurs then there's no error so test RBKUP FLAG
   ;       low by issuing a DECODE instruction with CSR bits 9 and 4 set.
   ;    5) Having cleared RBKUP FLAG it can be tested by issuing a NOP with
   ;       the skip control set to skip on REG BACKUP MASK FLAG. Check for
   ;       no skip.
   ;
   ; Error:
   ;
   ;    Error 1 - RBKUP FLAG failed high or skip logic failed.
   ;    Error 2 - RBKUP FLAG failed low or skip logic failed.
   ;
   ; Debug:
   ;
```

```
  ;     Error 1: If RBKUP FLAG failed high then either the signal itself is
```

```
   ;            stuck low or the logic causing the signal to be set is faulty
   ;            or the skip control logic is in error. The likely sources of
   ;            error are that RBKUP FLAG is stuck low or the logic that
   ;            produces the signal is in error. Useful places to check are
   ;            the CM RDEST PAL and SET RBKUP. If these are OK, then check
   ;            the RBKUP FLAG going into the MUX and INCR CTL PAL. This input
   ;            should be high, CSR 00-04 should be 1B(H), JUMP INSTR should
   ;            be high and the output on pin 12 should be low.
   ;    Error 2: If RBKUP FLAG failed low then the signal itself is stuck high
   ;            or the logic producing the signal is in error or the skip
   ;            control failed. Check the CM RDEST PAL and SET RBKUP and if
   ;            their not in error then RBKUP FLAG is probably stuck at zero.
   ;            If not, check the MUX and INCR CTL PAL. The inputs are the
   ;            same as error 1, except RBKUP FLAG should be low and outputs
   ;            12 and 19 should both be high.
   ;
   ;--
T.D:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.TD.0:
        JMP [WAIT.TD.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK.
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        JSR [ASSERT.NA]                 ;NO EXPECTED OR RECEIVED DATA
LOOP.TD.1:
;
;The following DECODE Instruction clears RBKUP FLAG
;
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/1,OPC.SPEC/1,JCTL/NO.JUMP.TST
        NOP
        MISC [SET.BACKUP.MASK.VALID]    ;SET FLAG
        SKIP.IF[BACKUP.MASK.VALID]      ;SKIP IF FLAG SET
        SKIP
        JMP [TD.2]                      ;GO ON IF OK
        MOV LS[ONES] TO WR[0]
        CLR WR[1]                       ;SET ERRONEOUS DATA TO FLAG ERROR
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TD.1]                 ;LOOP ON ERROR IF ENABLED
TD.2:
        INC LS[ERROR.NUMBER]            ;INC ERROR NUMBER TO 2
LOOP.TD.2:
;
;The following DECODE Instruction clears RBKUP FLAG
;
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0,IFUNC/1,OPC.SPEC/1,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[BACKUP.MASK.VALID]      ;SKIP IF FLAG SET
        JMP [END.TD]                    ;GO ON IF OK
        MOV LS[ONES] TO WR[0]
```

```
	CLR WR[1]			;SET ERRONEOUS DATA TO FLAG ERROR
	JSR [CHECK.RESULT]		;REPORT ERROR
	JMP [LOOP.TD.2]			;LOOP ON ERROR IF ENABLED
END.TD:
   .Page "TEST E - Branch False Test (M8390 CPU MODULE)"
   ;++
   ;
   ;
   ; Test Description:
   ;
   ;	This test checks the branch false condition. Branch false will be
   ;	asserted when IB 0 is asserted and when Pretest is not or when
   ;	Pretest is asserted and IB 0 is not. In order to test branch false
   ;	asserted when Pretest is not asserted, the IB BUS will be loaded
   ;	with 5D which puts a 1 on IB 0 and causes Pretest not to be asserted
   ;	If branch false is asserted it causes the signal, ENABLE IR ROM to
   ;	be asserted and in this way we're able to test if branch false is
   ;	asserted. The ENABLE IR ROM signal will enable the SPEC DECODE ROM
   ;	if the instruction is a DECODE instruction and the OPC/SPEC bit is
   ;	clear. This instruction will result in a jump to an address in WCS
   ;	contained in location 5D of the SPEC DECODE ROM depending on how the
   ;	upper bits are defined by the DECODE instruction. At that location
   ;	will be a jump to go back to the last instruction plus two. The
   ;	instruction before this instruction jumps to report the error.This
   ;	same procedure must be followed with IB equal to 5E which causes
   ;	branch false not to be asserted, the only difference being that the
   ;	Error  report is at the location jumped to from the DECODE instruction
   ;	instead of the instruction following the DECODE instruction. If there
   ;	is no jump then the branch false condition isn't stuck high or low.
   ;	As mentioned earlier BRANCH FALSE can be asserted if IB 0 SAVE is low
   ;	and PRETEST is low. In order to achieve this condition IB 0 can be
   ;	loaded with 1 and the condition codes can be defined by issuing a MOV
   ;	instruction to load LS[7F] with the condition codes that will set
   ;	BRANCH FALSE. A DECODE instruction with 5C on the IB BUS should cause
   ;	BRANCH FALSE to be asserted. The jump control field should be set for
   ;	BRANCH FALSE. IF there is a jump then another pattern of condition
   ;	codes that is known to assert BRANCH FALSE is loaded into LS[7F] and
   ;	a jump instruction or a NOP instruction with the skip control field
   ;	set is issued to test BRANCH FALSE. This pattern of testing is contin-
   ;	ued until all the conditions that cuase PRETEST to be asserted have
   ;	been tested.
   ;
   ; Assumptions:
   ;
   ;	It is assumed that the SPEC DECODE ROM and all the enabling signals
   ;	have been checked and it's assumed that the IB BUS has also been
   ;	checked and works.
   ;
   ; Test Steps:
   ;
   ;	1) Issue a MOV instruction to initialize LS[7E] to 0F. Issue a DECODE
   ;	   instruction with 5D on the IB BUS, which will cause a jump to some-
   ;	   place in the WCS.
   ;	2) If no jump then there's an error, issue error1.
   ;	3) If jump then there's no error then issue a DECODE instruction with
   ;	   5E on the IB BUS which should not cause a jump.
   ;	4) If no jump then no error.
```

```
;       5) If jump then  there's an error, issue error2.
;    6) Issue a MOV instruction to set PSL Z.
;    7) Issue a DECODE instruction to set the DATA TYPE to 3 and the CC
;       CLASS to one. If no jump then issue error3.
;    8) Issue a MOV instruction to set PSL N.
;    9) Issue a JUMP instruction to jump on BRANCH FALSE. If no jump then
;       issue error4.
;      10) Issue a MOV instruction to set PSL N and clear PSL V.
;      11) Issue a NOP instruction with a skip on BRANCH FALSE. If no skip
;       then issue error5.
;      12) Issue a MOV instruction to set PSL V and clear PSL N.
;      13) Issue a JUMP instruction to jump on BRANCH FALSE. If no jump then
;       issue error6.
;      14) Issue a MOV instruction to set PSL C.
;      15) Issue a DECODE instruction to set the DATA TYPE to 3 and the CC
;       CLASS to 1 with a jump on BRANCH FALSE in the jump control field.
;       If no jump then issue error7.
;      16) Issue a MOV instruction to set PSL V.
;      17) Issue a DECODE instruction to set the DATA TYPE to 3 and the CC
;       CLASS to 1 with a jump on BRANCH FALSE in the jump control field.
;       If no jump then issue error8.
;      18) Issue a MOV instruction to set PSL Z.
;      19) Issue a DECODE instruction to set the DATA TYPE to 3 and the CC
;       CLASS to 1 with a skip on BRANCH FALSE in the skip control field.
;       If skip then issue error9.
;      20) Issue a MOV instruction to clear PSL CC's.
;      21) Issue a DECODE instruction to set the DATA TYPE to 0 and the CC
;       CLASS to 0 with a skip on BRANCH FALSE in the skip control field.
;       If skip then issue errorA.
;
; Error:
;
;    Error 1 - BRANCH FALSE failed when it was asserted.
;    Error 2 - BRANCH FALSE failed when it was unasserted.
;    Error 3 - PRETEST failed.
;    Error 4 - PRETEST failed.
;    Error 5 - PRETEST failed.
;    Error 6 - PRETEST failed.
;    Error 7 - PRETEST failed.
;    Error 8 - PRETEST failed.
;    Error 9 - BRANCH FALSE was asserted.
;    Error A - BRANCH FALSE was asserted.
;
; Debug:
;
;    Error 1: BRANCH FALSE is low when it was expected to be high causing
;             an error. The likely sources of error are that the BRANCH
;             FALSE signal is stuck low or the BRANCH COND CC CLASS PAL
;            is in error.
;    Error 2: BRANCH FALSE is high when it was expected to be low causing
;             an error. The likely sources of error are that the BRANCH
;             FALSE signal is stuck high or the BRANCH COND CC CLASS PAL
;            is in error.
;    Error 3: BRANCH FALSE failed because PRETEST failed. PRETEST could have
;             failed as a result of failure in the BRANCH COND CC PAL or a
;             failure in PSL CC PAL. The important signal to look at is the
;            PSL Z signal.
;    Error 4: BRANCH FALSE failed because PRETEST failed. PRETEST probably
;               failed as a result of failure in the BRANCH COND CC PAL or
```

```
   ;            failure in the PSL CC PAL. The signal to examine here is the
   ;            PSL N signal.
   ;    Error 5: BRANCH FALSE failed because PRETEST failed. PRETEST probably
   ;            failed as a result of failure in the BRANCH COND CC PAL or
   ;            failure in the PSL CC PAL. The key signals here are the PSL N
   ;            and the PSL V signals.
   ;    Error 6: BRANCH FALSE failed because PRETEST failed. PRETEST probably
   ;            failed as a result of failure in the BRANCH COND CC PAL or
   ;            failure in the PSL CC PAL. The key signals here are the PSL N
   ;            and the PSL V signals.
   ;    Error 7: BRANCH FALSE failed because PRETEST failed. PRETEST probably
   ;            failed as a result of failure in the BRANCH COND CC PAL or
   ;            failure in the PSL CC PAL. The key signal here is PSL C.
   ;    Error 8: BRANCH FALSE failed because PRETEST failed. PRETEST failed
   ;            as a result of failure in the BRANCH COND CC PAL or failure
   ;            in the PSL CC PAL. The key signal here is PSL V.
   ;    Error 9: BRANCH FALSE is low when it was expected to be high causing
   ;            an error.  The likely sources of error are that the BRANCH
   ;            FALSE signal is stuck low or a failure in the BRANCH COND CC
   ;            PAL.  Examine PSL Z stuck low.
   ;    Error A: BRANCH FALSE was low (asserted) when it was expected to be
   ;            high (unasserted).   The most likely source of this error is
   ;            a failure in the BRANCH COND CC PAL where PSL Z is stuck high.
   ;
   ;--
T.E:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.TE.0:
        JMP [WAIT.TE.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK.
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        JSR [ASSERT.NA]                 ;NO EXPECTED OR RECIEVED DATA
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 94
        MOV LS[#1] TO WR[0]
        MOV WR[0] TO LS[T10]            ;T10 = 1
LOOP.TE.1:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        CLR LS[PC]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH 5D
        CLR LS[PSL.CC]                          ;CLEAR CONDITION CODES
;
;This DECODE will jump on BRANCH FALSE to a location in the 256 Block of inc's
;
        OPC2/DECODE,IB.REQ/1,BPC/NOP,DEC.ADRS/7,IFUNC/1,OPC.SPEC/1,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[BR.FALSE] TO [TE.2]      ;JUMP IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TE.1]                 ;LOOP ON ERROR IF ENABLED
TE.2:
        INC LS[ERROR.NUMBER]
LOOP.TE.2:
        CLR WR[0]
        MOV WR[0] TO LS[T10]            ;T10 = 0
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        CLR LS[PC]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH 5E
        CLR LS[PSL.CC]                  ;CLEAR CONDITION CODES
;
;This DECODE will jump on BRANCH FALSE to a location in the 256 Block of inc's
;
        OPC2/DECODE,IB.REQ/1,BPC/NOP,DEC.ADRS/7,IFUNC/1,OPC.SPEC/1,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[BR.FALSE]               ;SKIP IF ERROR
        JMP [TE.3]
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;CHECK THAT DECODE DID NOT JUMP
        JMP [LOOP.TE.2]                 ;LOOP ON ERROR IF ENABLED
TE.3:
        INC LS[ERROR.NUMBER]
        MOV LS[#10] TO WR[0]
        MOV WR[0] TO LS[T10]            ;T10 = 10
LOOP.TE.3:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH 10
        MOV LS[BIT2] TO WR[0]
        MOV WR[0] TO LS[PSL.CC]         ;SET PSL Z
;
;This DECODE sets DT to 0 & CC Class to 0
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[BR.FALSE] TO [TE.4]      ;JUMP IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TE.3]                 ;LOOP ON ERROR IF ENABLED
TE.4:
        INC LS[ERROR.NUMBER]
        MOV LS[#4] TO WR[0]
        MOV WR[0] TO LS[T10]            ;T10 = 4
LOOP.TE.4:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH 4
        MOV LS[BIT0] TO WR[0]
        MOV WR[0] TO LS[PSL.CC]         ;SET PSL C
;
```

```
;This DECODE sets DT to 0 & CC Class to 3
```

```
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[BR.FALSE] TO [TE.5]      ;JUMP IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TE.4]                 ;LOOP ON ERROR IF ENABLED
TE.5:
        INC LS[ERROR.NUMBER]
        MOV LS[#4] TO WR[0]
        ADD LS[#10] TO WR[0]
        MOV WR[0] TO LS[T10]            ;T10 = 14
LOOP.TE.5:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH 14
        MOV LS[BIT3] TO WR[0]
        MOV WR[0] TO LS[PSL.CC]         ;SET PSL N
;
;This DECODE sets DT to 1 & CC Class to 0
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[BR.FALSE] TO [TE.6]      ;JUMP IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TE.5]                 ;LOOP ON ERROR IF ENABLED
TE.6:
        INC LS[ERROR.NUMBER]
        MOV LS[#10] TO WR[0]
        SUB LS[#2] FROM WR[0]
        MOV WR[0] TO LS[T10]            ;T10 = E
LOOP.TE.6:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;FILL IB WITH E
        MOV LS[BIT1] TO WR[0]
        MOV WR[0] TO LS[PSL.CC]         ;SET PSL V
;
;This DECODE sets DT to 0 & CC Class to 2
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[BR.FALSE] TO [TE.7]      ;JUMP IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TE.6]                 ;LOOP ON ERROR IF ENABLED
TE.7:
        INC LS[ERROR.NUMBER]
        MOV LS[#8] TO WR[0]
        MOV WR[0] TO LS[T10]            ;T10 = 8
LOOP.TE.7:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
```

```
	MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]	;FILL IB WITH 8
```

```
        MOV LS[BIT3] TO WR[0]
        MOV WR[0] TO LS[PSL.CC]           ;SET PSL N (V CLEAR)
;
;This DECODE sets DT to 3 & CC Class to 1
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[BR.FALSE] TO [TE.8]        ;JUMP IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]                ;REPORT ERROR
        JMP [LOOP.TE.7]                   ;LOOP ON ERROR IF ENABLED
TE.8:
        INC LS[ERROR.NUMBER]
        MOV LS[#8] TO WR[0]
        MOV WR[0] TO LS[T10]              ;T10 = 8
LOOP.TE.8:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]        ;FILL IB WITH 8
        MOV LS[BIT1] TO WR[0]
        MOV WR[0] TO LS[PSL.CC]           ;SET PSL V (N CLEAR)
;
;This DECODE sets DT to 3 & CC Class to 1
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        JMP.IF[BR.FALSE] TO [TE.9]        ;JUMP IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]                ;REPORT ERROR
        JMP [LOOP.TE.8]                   ;LOOP ON ERROR IF ENABLED
TE.9:
        INC LS[ERROR.NUMBER]
        MOV LS[#10] TO WR[0]
        ADD LS[#1] TO WR[0]
        MOV WR[0] TO LS[T10]              ;T10 = 11
LOOP.TE.9:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]        ;FILL IB WITH 11
        MOV LS[BIT2] TO WR[0]
        MOV WR[0] TO LS[PSL.CC]           ;SET PSL Z
;
;This DECODE sets DT to 3 & CC Class to 1
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[BR.FALSE]                 ;SKIP IF ERROR
        JMP [TE.A]
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]                ;REPORT ERROR
        JMP [LOOP.TE.9]                   ;LOOP ON ERROR IF ENABLED
TE.A:
        INC LS[ERROR.NUMBER]
        MOV LS[#10] TO WR[0]
```

```
    MOV WR[0] TO LS[T10]            ;T10 = 10
```

```
LOOP.TE.A:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]       ;FILL IB WITH 10
        CLR WR[0]
        MOV WR[0] TO LS[PSL.CC]         ;CLEAR OUT CC'S
;
;This DECODE sets DT to 0 & CC Class to 0
;
        OPC2/DECODE,DEC.ADRS/7,IFUNC/1,IB.REQ/1,BPC/NOP,OPC.SPEC/CM.IRD,JCTL/NO.JUMP.TST
        NOP
        SKIP.IF[BR.FALSE]               ;SHOULD NOT SKIP
        JMP [END.TE]                    ;END OF TEST IF HERE
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TE.A]                 ;LOOP ON ERROR IF ENABLED
END.TE:
    .Page "TEST F - Jump Condition Test (M8390 CPU MODULE)"
    ;++
    ;
    ;
    ; Test Description:
    ;
    ;   This test checks that the miscellaneous jump conditions that haven't
    ;   been tested yet work. A condition can be jumped on by using either a
    ;   JUMP instruction or a DECODE instruction with the condition in the
    ;   JCTL field. The conditions that will be tested are IB VALID, a JUMP
    ;   on a DECODE instruction, JSR if IB VALID, and NO JUMP; SKIP if IB
    ;   VALID. IB VALID can be asserted by issuing a DECODE instruction that
    ;   causes a memory request. Once IB VALID is asserted the three conditions
    ;   depending on IB VALID can be tested and then the other unconditional
    ;   jump conditions can be tested.
    ;
    ; Assumptions:
    ;
    ;   It's assumed that the IB BUS and the SPEC DECODE ROM have been
    ;   checked and work. The IB VALID signal should have been tested
    ;   high.
    ;
    ; Test Steps:
    ;
    ;   1) Issue a DECODE instruction to to assert IB VALID with the JCTL
    ;      field set to IB VALID. If no jump then issue error1.
    ;   2) Issue a JUMP instruction to jump on IB VALID. If there is no
    ;      jump then issue error2.
    ;   3) Issue a JUMP instruction to JSR if IB VALID. If there is no JSR
    ;      then issue error3. If there is a JSR then issue a return.
    ;   4) Issue a JUMP instruction to NO JUMP; SKIP if IB VALID. If there
    ;      is no SKIP then issue error4.
    ;   5) Issue a DECODE instruction with the JCTL field set to JUMP. If
    ;      no jump then issue error5.
    ;   6) Issue a DECODE instruction that will set IB VALID with the JCTL
    ;      field set to JSR if IB VALID. If there is no JSR then issue error6.
    ;      If there is a JSR then issue a return.
    ;
    ; Error:
```

```
;
```

```
   ;    Error 1 - There was no jump on IB VALID with a DECODE instruction.
   ;    Error 2 - There was no jump on IB VALID with a JUMP instruction.
   ;    Error 3 - There was no JSR on IB VALID with a JUMP instruction.
   ;    Error 4 - There was no NO JUMP; SKIP on IB VALID with a JUMP instruction
   ;    Error 5 - There was no JUMP with a DECODE instruction.
   ;    Error 6 - There was no JSR on IB VALID with a DECODE instruction.
   ;
   ; Debug:
   ;
   ;    Error 1: The most likely source of error if JUMP ON IB VALID failed
   ;            is the SEQ CTL PAL. The logic concerning the IB VALID signal
   ;            has been tested previously and should be working.
   ;    Error 2: The most likely source of error is the SEQ CTL PAL.
   ;    Error 3: The most likely source of error is the SEQ CTL PAL.
   ;    Error 4: The most likely source of error is the SEQ CTL PAL.
   ;    Error 5: The most likely source of error is the SEQ CTL PAL.
   ;    Error 6: The most likely source of error is the SEQ CTL PAL.
   ;
   ;--
T.F:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.TF.0:
        JMP [WAIT.TF.0]                 ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        MOV LS[HI.IPL] TO WR[0]
        MOV WR[0] TO LS[PSL.HW]         ;CLEAR COMPAT MODE
        CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK.
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        JSR [ASSERT.NA]
        CLR LS[T10]
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 94
LOOP.TF.1:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;CLEAR MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;CLEAR IB
        JSR [TF.5.NOP]
        JSR [TF.DECODE.1]
        JMP [TF.2]                      ;GO ON IF RETURN
TF.DECODE.1:
        OPC2/DECODE,BPC/NOP,DEC.ADRS/7,IFUNC/0,OPC.SPEC/1,JCTL/JUMP.VALID
        NOP
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TF.1]                 ;LOOP ON ERROR IF ENABLED
TF.2:
        INC LS[ERROR.NUMBER]
LOOP.TF.2:
```

```
MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
```

```
        WRITE.MEM LS[T10]                    ;CLEAR MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]          ;CLEAR IB
        JSR [TF.5.NOP]
        JMP.IF[JUMP.VALID] TO [TF.3]     ;JUMP IF OK
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]                   ;REPORT ERROR
        JMP [LOOP.TF.2]                      ;LOOP ON ERROR IF ENABLED
TF.3:
        INC LS[ERROR.NUMBER]
LOOP.TF.3:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                    ;CLEAR MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]          ;CLEAR IB
        JSR [TF.5.NOP]
        CLR WR[3]
        JMP.IF[JSR.VALID] TO [TF.SUB3]
        TST WR[3],
         DT(LONG)&SET.ALU.CC                 ;WAS WR3 INCREMENTED?
        JMP.IF[NEQ] TO [END.TF]                     ;OK IF SO
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]                   ;REPORT ERROR
        JMP [LOOP.TF.3]                      ;LOOP ON ERROR IF ENABLED
TF.4:
        INC LS[ERROR.NUMBER]
LOOP.TF.4:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                    ;CLEAR MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]          ;CLEAR IB
        JSR [TF.5.NOP]
        JMP.IF[NO.JUMP.TST] TO [TF.4.ERR]           ;JUMP IF OK
        SKIP                                 ;SKIP IF NOSKIP
        JMP [TF.5]                           ;JUMP IF SKIPPED
TF.4.ERR:
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]                   ;REPORT ERROR
        JMP [LOOP.TF.4]                      ;LOOP ON ERROR IF ENABLED
TF.5:
        INC LS[ERROR.NUMBER]
LOOP.TF.5:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]                    ;CLEAR MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]          ;CLEAR IB
        JSR [TF.5.NOP]
        JSR [TF.DECODE.5]
        JMP [TF.6]                           ;GO ON IF RETURN
TF.DECODE.5:
        OPC2/DECODE,BPC/NOP,DEC.ADRS/7,IFUNC/0,OPC.SPEC/1,JCTL/JUMP
        NOP
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]                   ;REPORT ERROR
        JMP [LOOP.TF.5]                      ;LOOP ON ERROR IF ENABLED
TF.6:
        INC LS[ERROR.NUMBER]
```

```
LOOP.TF.6:
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;CLEAR MMLOC 94
        MEM.REQ[IB.FILL] ADRS[T8] DT[LONG]      ;CLEAR IB
        JSR [TF.5.NOP]
        CLR WR[3]
        OPC2/DECODE,BPC/NOP,DEC.ADRS/7,IFUNC/0,OPC.SPEC/1,JCTL/JSR.VALID
        NOP
        TST WR[3],
         DT(LONG)&SET.ALU.CC            ;WAS WR3 INCREMENTED?
        JMP.IF[NEQ] TO [END.TF]                 ;OK IF SO
        CLR WR[0]
        MOV LS[ONES] TO WR[1]
        JSR [CHECK.RESULT]              ;REPORT ERROR
        JMP [LOOP.TF.6]                 ;LOOP ON ERROR IF ENABLED
        JMP [END.TF]
TF.SUB3:
        INC WR[3]
        RETURN
TF.5.NOP:
        NOP                             ;ALLOW TIME FOR MEMORY TO FINISH
        NOP
        NOP
        NOP
        NOP
        RETURN
END.TF:
   .Page "TEST 10 - DEST CTL ROM And SRC.ALU CTL PAL Tests (M8390  CPU Module)"
   ;++
   ;
   ; Test Description:
   ;
   ;    This test will test every location in the DEST CTL ROM and all pos-
   ;    sible inputs to the SRC.ALU CTL PAL that were not tested in Test F of
   ;    the second module.  It is meant to verify that any u-code instuction
   ;    will control the 2901 as expected, even if that instruction has not
   ;    tested in previous tests. Some instructions already tested will not
   ;    be repeated here.  The test will be divided into sections correspond-
   ;    ing to the type of instruction being tested i.e. MEM.REQ, and DECODE.
   ;    The memory will be disabled throughout most of this test.  Only the
   ;    2901 modes will be checked.  It does not make sense to expand this
   ;    description further.  Instead refer to the comments in the listing.
   ;
   ; Assumptions:
   ;
   ;    It is assumed that all previous tests have run successfully.
   ;
   ; Test Steps:
   ;
   ;    See comments in listing.
   ;
   ; Errors:
   ;
   ;    OTHER Data : DEST CTL ROM Address
   ;
   ;    Error 1 - DECODE micro instruction error (BPC enabled)
   ;    Error 2 - DECODE micro instruction error (BPC disabled)
   ;    Error 3 - MOVE micro instruction error
```

```
;	Error 4 - MEM REQ micro instruction error
```

```
   ;
   ; Debug:
   ;
   ;    With the exception of error 3 with OTHER=C0, any and all errors will
   ;    involve either the DEST CTL ROM or the SRC.ALU CTL PAL.  The expected
   ;    outputs can be determined by referring to the 2901 control lines table
   ;    and the modes specified in the listing.  In this manner, the failure
   ;    can be traced to one of the two IC's listed above.
   ;
   ;    Error 3, OTHER=C0 - This error could indicate a problem with the LS
   ;    CONTROL PAL or the input CSR 17, 18, or 19.  Check these CSR inputs
   ;    for lows during the MOV MEM.DATA TO WR[1] XCHG TO LS[T9] instruction.
   ;    If these inputs are OK, the PAL is probably bad.
   ;
   ;
   ;--
T.10:
        MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT15 IN WR0 FOR CONTROL AND STATUS WORD
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT15 IN CNTROL AND STATUS WORD
                                        ; BIT15 INDICATES START OF TEST TO CONSOLE
        MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
WAIT.T10.0:
        JMP [WAIT.T10.0]                ;LOOP HERE WAITING FOR CONSOLE TO RESPOND
        CLR LS[ERROR.MASK]              ;CLEAR ERROR MASK.
        CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER
        MOV LS[#1] TO WR[0]             ;SET WR0 TO 1
        MOV WR[0] TO LS[ERROR.NUMBER]   ;SET ERROR NUMBER TO FIRST ERROR
        ROL WR[0]                       ;SET WR0 TO 2
        MOV WR[0] TO LS[MODULE.NUM]     ;SET UP MODULE CODE FOR CPU MODULE
        JSR [ASSERT.OTHER]              ;OTHER DATA FIELD WILL BE USED
        CLR LS[ADDRESS.DATA]            ;INIT "OTHER" DATA TO 0
;
; Decode Instruction Tests
;
LOOP.T10.1.00:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/00,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.00]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.01:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/04,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.01]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.02:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/08,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.02]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
```

```
LOOP.T10.1.03:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/0C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.03]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.04:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/10,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.04]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.05:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/14,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.05]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.06:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/18,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.06]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.07:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/1C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.07]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.08:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/20,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.08]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.09:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/24,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.09]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.0A:
        CLR LS[PC]                      ;SET PC TO 0
```

```
OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/28,IFUNC/1,JCTL/NO.JUMP.TST
```

```
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.0A]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.0B:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/2C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.0B]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.0C:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/30,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.0C]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.0D:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/34,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.0D]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.0E:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/38,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.0E]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.1.0F:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/3C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB1]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS WRITTEN TO WR4
        JMP [LOOP.T10.1.0F]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
T10.2:
        MOV LS[#8] TO WR[0]
        DEC WR[0]
        MOV WR[0] TO LS[PC]             ;PC = 7
;
; Write an 8 (PC+1) to WR4
;
        OPC2/DECODE,BPC/SAVE.PC,DEC.ADRS/3C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        INC LS[ERROR.NUMBER]            ;ERROR # = 2
LOOP.T10.2.10:
        CLR LS[PC]                      ;SET PC TO 0
```

```
OPC2/DECODE,BPC/NOP,DEC.ADRS/00,IFUNC/1,JCTL/NO.JUMP.TST
```

```
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.10]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.11:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/04,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.11]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.12:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/08,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.12]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.13:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/0C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.13]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.14:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/10,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.14]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.15:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/14,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.15]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.16:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/18,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.16]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.17:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/1C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.17]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.18:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/20,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.18]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.19:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/24,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.19]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.1A:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/28,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.1A]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.1B:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/2C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.1B]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.1C:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/30,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        NCP
        NOP
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.1C]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.1D:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/34,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.1D]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.1E:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/38,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.1E]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]            ;INC "OTHER" DATA FIELD
LOOP.T10.2.1F:
        CLR LS[PC]                      ;SET PC TO 0
        OPC2/DECODE,BPC/NOP,DEC.ADRS/3C,IFUNC/1,JCTL/NO.JUMP.TST
        NOP
        JSR [T10.SUB2]                  ;GO TO SUBROUTINE
        JSR [CHECK.RESULT]              ;CHECK THAT INCREMENTED PC WAS NOT WRITTEN TO WR4
        JMP [LOOP.T10.2.1F]             ;LOOP ON ERROR IF ENABLED
;
; Move Instruction Tests
;
T10.3:
        INC LS[ERROR.NUMBER]            ;ERROR # = 3
        MOV LS[#80] TO WR[0]
        ADD LS[#40] TO WR[0]
        MOV WR[0] TO LS[ADDRESS.DATA]   ;INIT "OTHER" DATA TO C0
LOOP.T10.3.C0:
        CLR LS[WR.BACKUP]               ;CLEAR LS WR.BACKUP TO START
        MOV LS[ALTER.ODD] TO WR[0]      ;SET UP PATTERN IN WR0
        MOV LS[#20] TO WR[2]
        MOV WR[2] TO LS[T8]             ;T8 = 20
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]           ;WRITE TO WR0
        MOV LS[#40] TO WR[1]            ;EXPECTED = CONTENTS OF MMLOC 20
        JSR [CHECK.RESULT]              ;CHECK MOV.MEM.WR
        JMP [LOOP.T10.3.C0]             ;LOOP ON ERROR IF ENABLED
        MOV LS[WR.BACKUP] TO WR[0]      ;GET BACKUP
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTED = ORIGINAL WR0
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS SAVED IN LS 11
        JMP [LOOP.T10.3.C0]             ;LOOP ON ERROR IF ENABLED
        MOV LS[ADDRESS.DATA] TO WR[0]
        ADD LS[#8] TO WR[0]
        MOV WR[0] TO LS[ADDRESS.DATA]   ;UPDATE "OTHER" DATA
        MOV LS[#80] TO WR[0]
        ADD LS[#10] TO WR[0]
        ADD LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 94
LOOP.T10.3.C8:
        MOV LS[ALTER.ODD] TO WR[0]
        MOV WR[0] TO LS[T10]            ;WRITE PATTERN TO LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[T10]               ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]           ;READ BACK MMLOC 94
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTED = AAAAAAAA
        JSR [CHECK.RESULT]              ;CHECK MOV.LS.MEM
        JMP [LOOP.T10.3.C8]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.C9:
        MOV LS[ALTER.ODD] TO WR[0]
        MOV WR[0] TO LS[L31]            ;WRITE PATTERN TO LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[L31]               ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]           ;READ BACK MMLOC 94
MOV LS[ALTER.ODD] TO WR[1]       ;EXPECTED = AAAAAAAA
```

```
        JSR [CHECK.RESULT]                      ;CHECK MOV.LS.MEM
        JMP [LOOP.T10.3.C9]                     ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.CA:
        MOV LS[ALTER.ODD] TO WR[0]
        MOV WR[0] TO LS[L53]                    ;WRITE PATTERN TO LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[L53]                       ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]                   ;READ BACK MMLOC 94
        MOV LS[ALTER.ODD] TO WR[1]              ;EXPECTED = AAAAAAAA
        JSR [CHECK.RESULT]                      ;CHECK MOV.LS.MEM
        JMP [LOOP.T10.3.CA]                     ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.CB:
        MOV LS[ALTER.ODD] TO WR[0]
        MOV WR[0] TO LS[L73]                    ;WRITE PATTERN TO LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[L73]                       ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]                   ;READ BACK MMLOC 94
        MOV LS[ALTER.ODD] TO WR[1]              ;EXPECTED = AAAAAAAA
        JSR [CHECK.RESULT]                      ;CHECK MOV.LS.MEM
        JMP [LOOP.T10.3.CB]                     ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.CC:
        MOV LS[ALTER.ODD] TO WR[0]
        MOV WR[0] TO LS[L90]                    ;WRITE PATTERN TO LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[L90]                       ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]                   ;READ BACK MMLOC 94
        MOV LS[ALTER.ODD] TO WR[1]              ;EXPECTED = AAAAAAAA
        JSR [CHECK.RESULT]                      ;CHECK MOV.LS.MEM
        JMP [LOOP.T10.3.CC]                     ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.CD:
        MOV LS[ALTER.ODD] TO WR[0]
        MOV WR[0] TO LS[LB0]                    ;WRITE PATTERN TO LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[LB0]                       ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]                   ;READ BACK MMLOC 94
        MOV LS[ALTER.ODD] TO WR[1]              ;EXPECTED = AAAAAAAA
        JSR [CHECK.RESULT]                      ;CHECK MOV.LS.MEM
        JMP [LOOP.T10.3.CD]                     ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.CE:
        MOV LS[ALTER.ODD] TO WR[0]
        MOV WR[0] TO LS[LD0]                    ;WRITE PATTERN TO LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[LD0]                       ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]                   ;READ BACK MMLOC 94
        MOV LS[ALTER.ODD] TO WR[1]              ;EXPECTED = AAAAAAAA
        JSR [CHECK.RESULT]                      ;CHECK MOV.LS.MEM
        JMP [LOOP.T10.3.CE]                     ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
```

```
LOOP.T10.3.CF:
        MOV LS[ALTER.ODD] TO WR[0]
        MOV WR[0] TO LS[LF0]            ;WRITE PATTERN TO LS LOCATION
        MEM.REQ[WRITE.P] ADRS[T8] DT[LONG]
        WRITE.MEM LS[LF0]               ;WRITE PATTERN TO MMLOC 94
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO WR[0]           ;READ BACK MMLOC 94
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTED = AAAAAAAA
        JSR [CHECK.RESULT]              ;CHECK MOV.LS.MEM
        JMP [LOOP.T10.3.CF]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
        MOV LS[ADDRESS.DATA] TO WR[0]
        ADD LS[#10] TO WR[0]
        MOV WR[0] TO LS[ADDRESS.DATA]   ;UPDATE "OTHER DATA" TO E0
        MOV LS[#4] TO WR[0]
        MOV WR[0] TO LS[T8]             ;T8 = 4
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[T11]
LOOP.T10.3.E0:
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[T10]         ;WRITE TO LS LOCATION
        MOV LS[T10] TO WR[0]            ;GET RECEIVED DATA
        MOV LS[T11] TO WR[1]            ;EXPECTED = 00FF00FF
        JSR [CHECK.RESULT]              ;CHECK THAT LS WAS WRITTEN CORRECTLY
        JMP [LOOP.T10.3.E0]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.E1:
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[L31]         ;WRITE TO LS LOCATION
        MOV LS[L31] TO WR[0]            ;GET RECEIVED DATA
        MOV LS[T11] TO WR[1]            ;EXPECTED = 00FF00FF
        JSR [CHECK.RESULT]              ;CHECK THAT LS WAS WRITTEN CORRECTLY
        JMP [LOOP.T10.3.E1]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.E2:
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[L53]         ;WRITE TO LS LOCATION
        MOV LS[L53] TO WR[0]            ;GET RECEIVED DATA
        MOV LS[T11] TO WR[1]            ;EXPECTED = 00FF00FF
        JSR [CHECK.RESULT]              ;CHECK THAT LS WAS WRITTEN CORRECTLY
        JMP [LOOP.T10.3.E2]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.E3:
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[L73]         ;WRITE TO LS LOCATION
        MOV LS[L73] TO WR[0]            ;GET RECEIVED DATA
        MOV LS[T11] TO WR[1]            ;EXPECTED = 00FF00FF
        JSR [CHECK.RESULT]              ;CHECK THAT LS WAS WRITTEN CORRECTLY
        JMP [LOOP.T10.3.E3]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.E4:
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[L90]         ;WRITE TO LS LOCATION
        MOV LS[L90] TO WR[0]            ;GET RECEIVED DATA
        MOV LS[T11] TO WR[1]            ;EXPECTED = 00FF00FF
        JSR [CHECK.RESULT]              ;CHECK THAT LS WAS WRITTEN CORRECTLY
        JMP [LOOP.T10.3.E4]             ;LOOP ON ERROR IF ENABLED
```

```
INC LS[ADDRESS.DATA]
```

```
LOOP.T10.3.E5:
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[LB0]         ;WRITE TO LS LOCATION
        MOV LS[LB0] TO WR[0]            ;GET RECEIVED DATA
        MOV LS[T11] TO WR[1]            ;EXPECTED = 00FF00FF
        JSR [CHECK.RESULT]              ;CHECK THAT LS WAS WRITTEN CORRECTLY
        JMP [LOOP.T10.3.E5]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.E6:
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[LD0]         ;WRITE TO LS LOCATION
        MOV LS[LD0] TO WR[0]            ;GET RECEIVED DATA
        MOV LS[T11] TO WR[1]            ;EXPECTED = 00FF00FF
        JSR [CHECK.RESULT]              ;CHECK THAT LS WAS WRITTEN CORRECTLY
        JMP [LOOP.T10.3.E6]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.E7:
        MEM.REQ[READ.P] ADRS[T8] DT[LONG]
        MOV MEM.DATA TO LS[LF0]         ;WRITE TO LS LOCATION
        MOV LS[LF0] TO WR[0]            ;GET RECEIVED DATA
        MOV LS[T11] TO WR[1]            ;EXPECTED = 00FF00FF
        JSR [CHECK.RESULT]              ;CHECK THAT LS WAS WRITTEN CORRECTLY
        JMP [LOOP.T10.3.E7]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.E8:
        MOV LS[ALTER.ODD] TO WR[0]      ;WRITE PATTERN TO WR[0]
        MOV FPA TO LS[T10]
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTD = WR0 UNCHANGED
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS NOT CHANGED
        JMP [LOOP.T10.3.E8]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.E9:
        MOV LS[ALTER.ODD] TO WR[0]      ;WRITE PATTERN TO WR[0]
        MOV FPA TO LS[L31]
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTD = WR0 UNCHANGED
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS NOT CHANGED
        JMP [LOOP.T10.3.E9]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.EA:
        MOV LS[ALTER.ODD] TO WR[0]      ;WRITE PATTERN TO WR[0]
        MOV FPA TO LS[L53]
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTD = WR0 UNCHANGED
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS NOT CHANGED
        JMP [LOOP.T10.3.EA]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.EB:
        MOV LS[ALTER.ODD] TO WR[0]      ;WRITE PATTERN TO WR[0]
        MOV FPA TO LS[L73]
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTD = WR0 UNCHANGED
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS NOT CHANGED
        JMP [LOOP.T10.3.EB]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.EC:
        MOV LS[ALTER.ODD] TO WR[0]      ;WRITE PATTERN TO WR[0]
        MOV FPA TO LS[L90]
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTD = WR0 UNCHANGED
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS NOT CHANGED
```

```
	JMP [LOOP.T10.3.EC]		;LOOP ON ERROR IF ENABLED
```

```
        INC LS[ADDRESS.DATA]
LOOP.T10.3.ED:
        MOV LS[ALTER.ODD] TO WR[0]      ;WRITE PATTERN TO WR[0]
        MOV FPA TO LS[LB0]
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTD = WR0 UNCHANGED
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS NOT CHANGED
        JMP [LOOP.T10.3.ED]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.EE:
        MOV LS[ALTER.ODD] TO WR[0]      ;WRITE PATTERN TO WR[0]
        MOV FPA TO LS[LD0]
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTD = WR0 UNCHANGED
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS NOT CHANGED
        JMP [LOOP.T10.3.EE]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.3.EF:
        MOV LS[ALTER.ODD] TO WR[0]      ;WRITE PATTERN TO WR[0]
        MOV FPA TO LS[LF0]
        MOV LS[ALTER.ODD] TO WR[1]      ;EXPECTD = WR0 UNCHANGED
        JSR [CHECK.RESULT]              ;CHECK THAT WR0 WAS NOT CHANGED
        JMP [LOOP.T10.3.EF]             ;LOOP ON ERROR IF ENABLED
;
; Mem Req Instruction Tests
;
T10.4:
        MOV LS[BIT16] TO WR[0]
        BIS LS[BIT22] TO WR[0]          ;SET CSR TO DISABLE MEMORY REFERENCES
        MOV WR[0] TO LS[CONTROL.STATUS] ;DISABLE MEMORY REFERENCES
        MISC [SET.CP.ATTN]              ;CALL CONSOLE
WAIT.T10.4:
        JMP [WAIT.T10.4]                ;WAIT FOR CONSOLE TO RESPOND
        INC LS[ERROR.NUMBER]            ;ERROR # = 4
        MOV LS[#40] TO WR[0]
        ADD LS[#20] TO WR[0]
        MOV WR[0] TO LS[ADDRESS.DATA]   ;"OTHER DATA" = 60
        MOV LS[#100] TO WR[0]
        MOV WR[0] TO LS[T10]            ;SET UP LS LOCATIONS
        MOV WR[0] TO LS[L31]
        MOV WR[0] TO LS[L53]
        MOV WR[0] TO LS[L73]
        MOV LS[#100] TO WR[1]           ;EXPECTED = 100
LOOP.T10.4.60:
        MEM.REQ[READ.P] ADRS[T10] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.60]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.61:
        MEM.REQ[READ.P] ADRS[L31] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.61]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.62:
        MEM.REQ[READ.P] ADRS[L53] DT[LONG]
```

```
MOV XWR[VAR] TO Q
```

```
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.62]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.63:
        MEM.REQ[READ.P] ADRS[L73] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.63]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.64:
        MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.64]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.65:
        MEM.REQ[WRITE.P] ADRS[L31] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.65]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.66:
        MEM.REQ[WRITE.P] ADRS[L53] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.66]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.67:
        MEM.REQ[WRITE.P] ADRS[L73] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.67]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.68:
        MEM.REQ[READ.V.NOCHK] ADRS[T10] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.68]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.69:
        MEM.REQ[READ.V.NOCHK] ADRS[L31] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.69]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.6A:
        MEM.REQ[READ.V.NOCHK] ADRS[L53] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.6A]                ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.6B:
        MEM.REQ[READ.V.NOCHK] ADRS[L73] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                     ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                 ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.6B]                ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.6C:
        MEM.REQ[WRITE.V.NOCHK] ADRS[T10] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                     ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                 ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.6C]                ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.6D:
        MEM.REQ[WRITE.V.NOCHK] ADRS[L31] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                     ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                 ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.6D]                ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.6E:
        MEM.REQ[WRITE.V.NOCHK] ADRS[L53] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                     ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                 ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.6E]                ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.6F:
        MEM.REQ[WRITE.V.NOCHK] ADRS[L73] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                     ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                 ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.6F]                ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.70:
        MEM.REQ[OCTA.WRITE.P] ADRS[T10] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                     ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                 ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.70]                ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.71:
        MEM.REQ[OCTA.WRITE.P] ADRS[L31] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                     ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                 ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.71]                ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.72:
        MEM.REQ[OCTA.WRITE.P] ADRS[L53] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                     ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]                 ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.72]                ;LOOP ON ERROR IF ENABLED
```

```
	INC LS[ADDRESS.DATA]
```

```
LOOP.T10.4.73:
        MEM.REQ[OCTA.WRITE.P] ADRS[L73] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.73]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.74:
        MEM.REQ[READ.MAINT.ADRS] ADRS[T10] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.74]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.75:
        MEM.REQ[READ.MAINT.ADRS] ADRS[L31] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.75]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.76:
        MEM.REQ[READ.MAINT.ADRS] ADRS[L53] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.76]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.77:
        MEM.REQ[READ.MAINT.ADRS] ADRS[L73] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.77]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.78:
        MEM.REQ[OCTA.READ.P] ADRS[T10] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.78]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.79:
        MEM.REQ[OCTA.READ.P] ADRS[L31] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.79]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.7A:
        MEM.REQ[OCTA.READ.P] ADRS[L53] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.7A]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.7B:
```

```
MEM.REQ[OCTA.READ.P] ADRS[L73] DT[LONG]
```

```
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.7B]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.7C:
        MEM.REQ[ISSUE.BG] ADRS[T10] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.7C]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.7D:
        MEM.REQ[ISSUE.BG] ADRS[L31] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.7D]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.7E:
        MEM.REQ[ISSUE.BG] ADRS[L53] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.7E]             ;LOOP ON ERROR IF ENABLED
        INC LS[ADDRESS.DATA]
LOOP.T10.4.7F:
        MEM.REQ[ISSUE.BG] ADRS[L73] DT[LONG]
        MOV XWR[VAR] TO Q
        MOV Q TO WR[0]                  ;GET VIRTUAL ADDRESS FROM WR5
        JSR [CHECK.RESULT]              ;CHECK THAT VIRTUAL ADD WAS WRITTEN TO WR5
        JMP [LOOP.T10.4.7F]             ;LOOP ON ERROR IF ENABLED
CLEANUP.T10:
        MOV LS[BIT16] TO WR[0]
        ROL WR[0]
        MOV WR[0] TO LS[L31]            ;RESTORE LOCATION 31
        MOV LS[BIT16] TO WR[0]
        MOV WR[0] TO LS[CONTROL.STATUS] ;RE-ENABLE MEMORY REQ
        MISC [SET.CP.ATTN]              ;CALL CONSOLE
WAIT.T10.CLNUP:
        JMP [WAIT.T10.CLNUP]            ;WAIT FOR CONSOLE TO RESPOND
        JMP [END.T10]                   ;END OF TEST
T10.SUB1:
        MOV XWR[BPC] TO Q               ;GET WR4
        MOV Q TO WR[0]
        MOV LS[#1] TO WR[1]             ;EXPECTED = INCREMENTED PC
        RETURN
T10.SUB2:
        MOV XWR[BPC] TO Q               ;GET WR4
        MOV Q TO WR[0]
        MOV LS[#8] TO WR[1]             ;EXPECTED = WR4 NOT WRITTEN
        RETURN
END.T10:
.PAGE
END.SECTION:
        MOV LS[BIT14] TO WR[0]          ;SET UP WR 0 FOR END OF SECTION
        MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL AND STATUS WORD TO
```

```
                                        ; REPORT END OF SECTION
```

```
        MISC [SET.CP.ATTN]                  ;ISSUE CPU ATTN TO CONSOLE
WAIT.EOS:
        JMP [WAIT.EOS]                      ;LOOP HERE WAITING FOR CONSOLE TO
                                            ; RESPOND
```